

Evaluación de métricas de calidad del software sobre un programa Java.

Proyecto de Fin de Máster en
Programación y Tecnología Software

Curso 2009-2010



Máster en Investigación en Informática.

Facultad de Informática.

Universidad Complutense de Madrid.



Autor:

Ana María García Sánchez.

Dirigido por:

Dtor. Manuel García Clavel.

Dpto. de Sistemas Informáticos y Computación.

Autorización

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: “Evaluación de métricas de calidad del software sobre un programa Java”, realizado durante el curso académico 2009-2010 bajo la dirección del Prof. Manuel García Clavel en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Firmado:

Ana María García Sánchez.

*“Cuando puedas medir lo que estás diciendo y expresarlo en números, sabrás algo acerca de eso;
pero cuando no puedes medirlo, cuando no puedes expresarlo en números, tus conocimientos
serán escasos y no satisfactorios”*

Lord Kelvin

“Lo que no sea medible, hazlo medible”

Galileo Galilei

“No se puede controlar lo que no se puede medir”

Tom De Marco

“No se puede predecir lo que no se puede medir”

Norman Fenton

Agradecimientos

En primer lugar, me gustaría dar las gracias al director del proyecto Manuel García Clavel por la idea inicial y por la ayuda recibida en cada uno de los objetivos propuestos, a Marina Egea por la revisión final del presente documento y muy especialmente, me gustaría agradecer a Miguel Ángel García toda la ayuda que me ha proporcionado a nivel técnico en todas las tecnologías utilizadas, sus explicaciones y soluciones han sido imprescindible para resolver todos los problemas, así como para conseguir todos los objetivos propuestos.

A Pablo, por su constante apoyo y ayuda, especialmente durante estos meses en los que hemos estado trabajando para sacar adelante nuestros respectivos proyectos.

Por último, quiero dar las gracias a mis amigos con los que he compartido estos años en la Universidad Complutense y muy especialmente quiero agradecer el apoyo de mi familia, mis hermanas y mis padres, cuyo esfuerzo y trabajo diario ha sido imprescindible para conseguir todos mis retos, como es el caso del Máster en Investigación en Informática y en concreto, este proyecto.

Índice de contenidos

RESUMEN DEL PROYECTO	VII
RESUMEN	VII
PALABRAS CLAVE.....	VII
ABSTRACT	IX
KEYWORDS.....	IX
APPROACH	X
1. INTRODUCCIÓN	1
1.1 METODOLOGÍA	1
1.2 OBJETIVOS DEL PROYECTO.....	1
<i>Fase I: Estado del arte</i>	<i>2</i>
<i>Fase II: Metamodelo de Java</i>	<i>2</i>
<i>Fase III: Representación de métricas.</i>	<i>3</i>
<i>Fase IV: Generación de código.</i>	<i>3</i>
<i>Fase V: Evaluación de métricas.</i>	<i>3</i>
1.3 COMPONENTES UTILIZADOS.....	3
1.3.1 Metamodelo de Java.....	3
1.3.2 SpoonEMF	4
1.3.3 JET Java Emitter Template.....	5
1.3.4 OCL Object Constraint Language.....	5
1.3.5 EOS A Java component for OCL evaluation	6
2. ESTADO DEL ARTE.....	7
2.1 MÉTRICAS DE CALIDAD DEL SOFTWARE	7
2.1.1 Métricas CK Chidamber y Kemerer	9
2.1.2 Métricas de Lorenz y Kidd (1994).....	11
2.1.3 Métricas R. Martin (1994).....	12
2.1.4 Métricas de Li-Henry.....	12
2.2 HERRAMIENTAS DE EVALUACIÓN DE MÉTRICAS PARA PROGRAMAS JAVA.....	12
2.2.1 JCSC – Java Coding Standard Checker.....	12
2.2.2 CheckStyle	14
2.2.3 JavaNCSS.....	15
2.2.4 JMT	17
2.2.5 Metrics Eclipse plugin	19
2.2.6 RSM Resource Standard Metrics.....	21
2.2.7 SDMetrics.....	22
2.2.8 SONAR.....	23
2.2.9 Kemis “Kybele Environment Measurement Information System”	25
3. DISEÑO TÉCNICO	27
3.1 MÓDULO ES.UCM.SPOON.CLIENT	28
3.1.1 Descripción	28
3.1.2 Tipo.....	28
3.1.3 Estructura.....	28

3.1.4	Procesamiento.....	29
3.2	MÓDULO SPOONToEOS.....	30
3.2.1	Descripción.....	30
3.2.2	Tipo.....	30
3.2.3	Estructura.....	30
3.2.4	Procesamiento.....	31
4.	RESULTADOS OBTENIDOS.....	51
4.1	PRUEBAS DE EJECUCIÓN	51
4.2	COMPARATIVA ENTRE HERRAMIENTAS	55
5.	CONCLUSIONES	57
5.1	OBJETIVOS CUMPLIDOS.....	57
5.2	TRABAJO FUTURO.....	58
APÉNDICE A. SPOONJDT		61
APÉNDICE B. CLIENTE PARA SPOONEMF Y LIMITACIONES.....		63
APÉNDICE C. PLANTILLA INICIAL MAIN.JET.....		69
APÉNDICE D. INSTALACIÓN Y EJECUCIÓN.....		71
BIBLIOGRAFÍA.....		73

Resumen del proyecto

Resumen

Este proyecto presenta los resultados de una investigación sobre la actual metodología de evaluación de la calidad del software. En particular incluye un resumen de las métricas más importantes que pueden encontrarse en la literatura para medir la calidad del software y de las herramientas existentes que, dándoles soporte, evalúan código fuente. Además, en el marco de este proyecto se ha implementado una herramienta cuya funcionalidad principal es evaluar métricas sobre programas Java y ofrecer al usuario el valor resultante de la evaluación cada una de las métricas a través de una interfaz de usuario.

En el primer capítulo se ofrece una visión general del proyecto, detallando los objetivos marcados y las fases por la que ha pasado el proyecto para realizar cada uno de los objetivos propuestos. Además, se describen los componentes externos que se han utilizado para el desarrollo de la herramienta de evaluación.

El segundo capítulo se centra en el estado del arte de este área de evaluación de la calidad del software. En él, se detallan las métricas que otros autores han especificado con anterioridad y algunas de las herramientas ya existentes que auditan y evalúan código fuente a partir de reglas y métricas ya definidas.

El tercer capítulo detalla el diseño técnico del proyecto, es decir, la arquitectura de la herramienta **spoonToEOS** y las funcionalidades que desempeña.

El cuarto capítulo presenta algunos de los resultados obtenidos con spoonToEOS y una comparación de esta herramienta con las herramientas que se referencian en el segundo capítulo. Esta comparación se resume en una tabla comparativa que muestra las diferencias entre ellas y las ventajas e inconvenientes que presentan cada una.

El quinto capítulo muestra las conclusiones que podemos sacar de este proyecto y el trabajo futuro que se podría realizar para ampliar la funcionalidad de la herramienta spoonToEOS.

Finalmente se incluyen algunos apéndices que especifican más detalladamente algún punto concreto del documento.

Palabras clave

Metamodelo, XML/XMI, SpoonEMF, JET Java Emitter Template, plugin, Object Constraint Language (OCL), componente EOS, métrica de calidad del software, evaluación código fuente.

Abstract

This project focuses on Software quality evaluation by obtaining the value of metrics which properly interpreted are indicators of possible flaws in the source code. We report on existing metrics and tools that measure software quality by evaluating the source code. Furthermore, we present a new tool to evaluate metrics on Java programs. The result of each metric evaluation together with a description of the metric is provided through a user interface.

The first chapter gives an overview of the project, detailing the objectives and project phases. We report also on the external components that are parts of the tool.

The second chapter focuses on the state of the art of this field of software engineering. It summarizes the metrics that can be found in literature and some of the existing tools that use rules and metrics to audit and evaluate source code.

The third chapter details the technical design of the project, i.e., how the tool **spoonToEOS** has been developed and which is its functionality.

The fourth chapter presents some of the results obtained applying the tool spoonToEOS to some projects of example and a comparison of this tool to others that are referenced in the second chapter. This comparison is summarized in a table to show the differences between them and the functionality they support.

The fifth chapter draws some conclusions and outlines future work that could enhance the functionality of the tool spoonToEOS.

Finally, we include some appendixes detailing the specification of the novel parts of the tool developed during this project.

Keywords

Metamodel, XML/XMI, SpoonEMF, JET Java Emitter Template, plugin, Object Constraint Language (OCL), EOS, software quality metric, source code evaluation.

Approach

We follow a model driven architecture (MDA) approach to obtain the values of metrics on Java programs. Our methodology can be summarized in four steps:

1. We provide an XML file (easily modifiable or extendable by the user) containing different metrics definition using the OCL language within the context of the Java metamodel.
2. We reverse engineer the Java program using the tool spoonEMF to get a Java metamodel instance as an XMI file.
3. We load the Java metamodel as a class diagram and the XMI file containing the instance as its object diagram into the EOS component.
4. We allow the user to choose the metrics to be evaluated by the EOS component on the instance file through a graphical user interface where (s)he can also inspect the results.

1. Introducción

En este primer apartado, se ofrece una introducción al proyecto realizado, la metodología utilizada y una descripción de los conceptos básicos que se deben tener para la comprensión del sistema.

1.1 Metodología

En este proyecto se sigue una metodología de arquitectura dirigida por modelos (MDA) para obtener los valores de las métricas sobre los programas Java. Podemos resumir nuestra metodología en cuatro pasos:

1. Proporcionamos un archivo XML (fácilmente modificable o extensible por el usuario) que contiene la definición de distintas métricas en el lenguaje de restricciones a objetos (OCL) usado en el contexto de un metamodelo construido para el lenguaje Java.
2. Realizamos ingeniería inversa de cualquier programa Java usando para ello la herramienta spoonEMF. El resultado es una instancia del metamodelo de Java guardada en un archivo XML.
3. Cargamos el metamodelo de Java como un diagrama de clases y el archivo XML que contiene la instancia como su diagrama de objetos en la componente EOS.
4. Permitimos al usuario escoger las métricas que quiera que la componente EOS evalúe sobre el archivo instancia del programa Java a través de una interfaz de usuario en la que también puede ver los resultados de dicha evaluación.

1.2 Objetivos del proyecto

El objetivo principal del proyecto es desarrollar un sistema de ejecución de métricas de calidad del software sobre el código fuente de una aplicación Java.

El proyecto se puede dividir en distintas fases, cada una de las fases está dirigida por ciertos objetivos concretos cuya consecución necesita de la investigación y el desarrollo de una o varias funcionalidades del sistema.

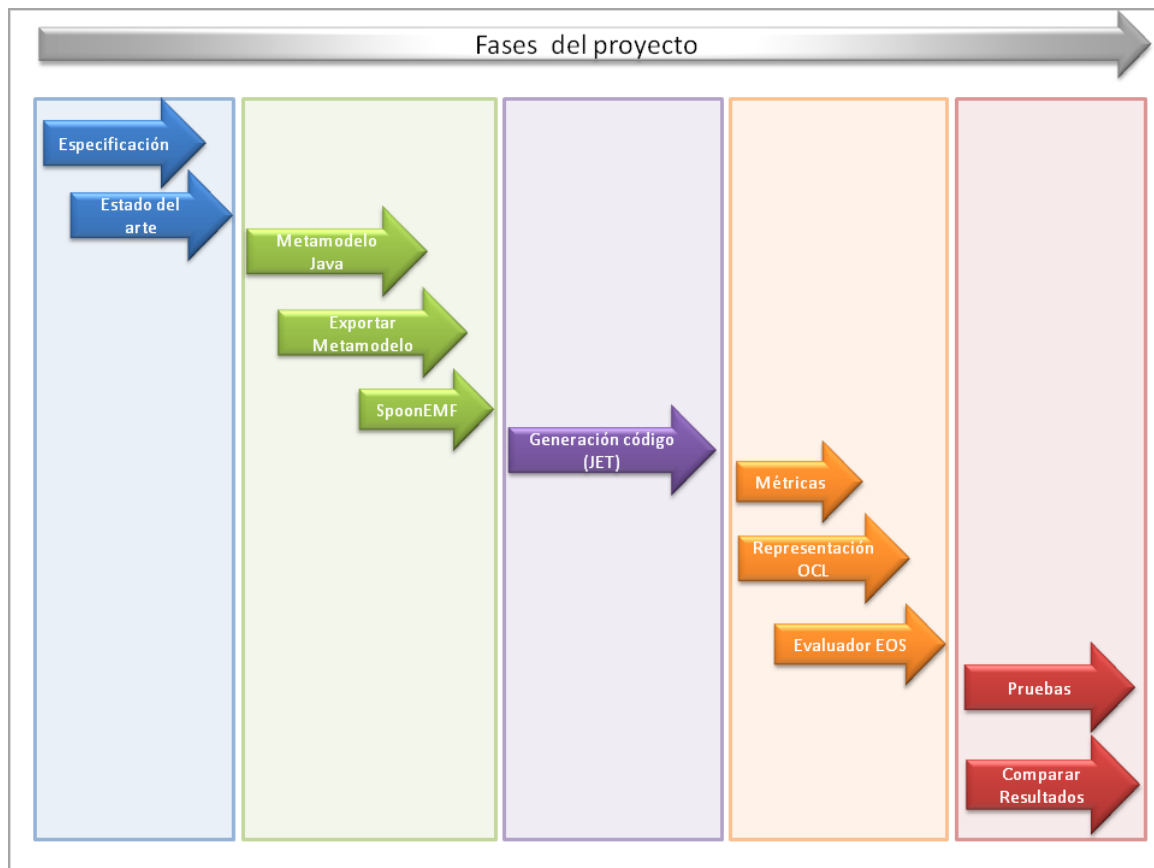


Ilustración 1: Objetivos: Fases del proyecto.

Fase I: Estado del arte

En la primera fase, se investigó el estado del arte de las métricas de calidad del software, recogiendo una lista de las métricas más utilizadas para medir la calidad del código fuente de un sistema concreto. Además se investigaron las herramientas existentes que sirven para evaluar este tipo de métricas, como por ejemplo: RSM “Resource Standard Metrics”, SONAR, SDMetrics. En el apartado [Estado del arte](#) se puede encontrar una breve descripción de cada una de ellas.

Fase II: Metamodelo de Java

A continuación, se estudió el metamodelo de Java propuesto por el Object Management Group (OMG) [1] para entender la interpretación como elementos de modelado de las constructos que pueden incluirse en un determinado programa Java: paquetes, clases, atributos, herencia, etcétera.

A partir de un determinado proyecto Java, se necesita obtener su modelo como instancia del metamodelo de Java para su posterior procesamiento. Para ello, se investigó la posible forma de obtener el metamodelo de un determinado programa en un fichero con formato XML, XMI, Ecore, etcétera. Finalmente se decidió utilizar SpoonEMF para generar una instancia del metamodelo de Java a partir de un determinado programa Java. El apartado [SpoonEMF](#) describe la funcionalidad de esta librería.

Fase III: Representación de métricas.

Para la representación de métricas se utiliza OCL. Las métricas obtenidas en la fase I que se han añadido al sistema, se especifican mediante expresiones OCL sobre el metamodelo de Java. Estas métricas son guardadas en un archivo XML. El apartado [OCL Object Constraint Language](#) muestra la librería de métricas resultante y el esquema que describe la estructura del documento XML.

Fase IV: Generación de código.

Una vez obtenido el modelo que corresponde al programa Java que queremos evaluar, necesitamos una forma de evaluar las métricas sobre él. Las métricas son escritas como funciones y consultas OCL y para su evaluación se ha utilizado el componente EOS. Este componente necesita como entradas el metamodelo de Java como diagrama de clases y una instancia del metamodelo de Java como diagrama de objetos, que en este caso, corresponde al modelo del código Java que se desea evaluar. En el apartado [EOS: A Java component for OCL evaluation](#) se especifica la funcionalidad de esta componente.

El código que interpreta el metamodelo de Java y el modelo del programa Java, se genera utilizando el plugin de eclipse JET. Esto se debe a que el código fuente de la herramienta implementada no es estático, depende de la instancia del metamodelo que corresponde al programa que se desea evaluar y se genera a partir del metamodelo de Java, del fichero que contiene la instancia del metamodelo de java y del fichero que contiene las métricas de calidad del software. En el apartado [JET Java Emitter Template](#) se detalla la funcionalidad de esta parte.

Por tanto, durante esta tercera fase se investigó JET, se aprendió a utilizar su lenguaje ejecutando algunos ejemplos y se desarrolló el componente que genera el código que se necesita para evaluar las métricas definidas a partir de las distintas plantillas. En el apartado [Generación de código](#) del tercer capítulo “Diseño Técnico” se detalla la implementación de la generación de código automática de este proyecto.

Fase V: Evaluación de métricas.

Finalmente, una vez obtenido el modelo de un determinado proyecto Java y la representación de las métricas como expresiones OCL, el sistema utiliza el componente EOS para evaluar cada una de las métricas sobre la instancia del metamodelo de Java y muestra al usuario el valor resultante de la evaluación y una pequeña descripción de cada una de las métricas procesadas.

1.3 Componentes utilizados

Este apartado, describe brevemente cada uno de los componentes utilizados en el desarrollo de la aplicación, ofreciendo una visión general de cada uno de estos componentes y de las funcionalidades que desempeñan cada uno de ellos en la ejecución de la aplicación.

1.3.1 Metamodelo de Java

El metamodelo de java [1] se define como la conjunción de cinco diagramas:

- El *diagrama de contenidos de clases*, donde se incluyen los paquetes, las clases o interfaces, los atributos, los métodos, las excepciones, etcétera.
- El *diagrama de polimorfismo*, representa las relaciones entre clases padre y clases hija, es decir, especifica la relación existente entre una clase y la clase de la que hereda o la relación existente entre una clase y la interfaz que implementa.
- El *diagrama de tipos Java: Field, JavaParameter, ArrayType, y JavaClass*. Define los tipos de los posibles elementos Java.
- El *diagrama de tipos de datos*, este diagrama incluye los tipos de datos primitivos y otros tipos de datos que se incluyen para completar el metamodelo de Java.
- El *diagrama que muestra la factorización del atributo "name"* definido en la clase padre.

1.3.2 SpoonEMF

SpoonEMF es una herramienta desarrollada por Triskell Group, que transforma un determinado software desarrollado en Java en un modelo Eclipse/EMF [2]. SpoonEMF es la unión entre Spoon y EMF; por su parte, Spoon proporciona un completo metamodelo de Java donde cualquier elemento del programa (clases, atributos, métodos, expresiones,...) es accesible. SpoonEMF transforma un completo programa Java en un único fichero XML.

El sistema utiliza la herramienta SpoonEMF para obtener un fichero XML a partir de un determinado programa Java, este fichero define el programa Java como una instancia del metamodelo de Java.

Para generar el fichero XML con la instancia del metamodelo de Java a partir de un programa Java cualquiera se han barajado dos posibles soluciones: utilizar el plugin de eclipse SpoonJDT [<http://spoon.gforge.inria.fr/Spoon/HomePage>] o implementar un componente que simule el comportamiento de esta herramienta utilizando las mismas librerías. En el primer caso, basta con instalar el plugin SpoonJDT en Eclipse y configurar Spoon tal y como se muestra en el [Apéndice A](#).

SpoonJDT genera la instancia del metamodelo de Java en el fichero que indica el usuario a partir del programa Java seleccionado. La utilización de este plugin tiene el siguiente inconveniente: no genera correctamente el modelo de aquellos programas que instancian tipos de datos almacenados en librerías externas que no incluyen su código fuente. Ver el [Apéndice A](#) de este documento para obtener información más detallada.

Para evitar este problema, se ha tomado la decisión de implementar un componente inicial denominado **spoonClient** que recibe como entradas el nombre del fichero XML donde generar la instancia del metamodelo de Java y la ruta donde está almacenado el proyecto cuya instancia se desea generar. El componente spoonClient simplemente invoca a la clase principal de spoonEMF `main.Java2XMLHelper` pasando como argumentos el fichero XML de la salida y la ruta del proyecto Java.

Al ejecutar spoonClient se genera el fichero con la instancia del metamodelo de Java. En este caso, el problema de incluir en el metamodelo los tipos de datos almacenados en librerías externas que no incluyen su código fuente, sigue estando presente, pero con la ventaja de que el fichero de salida se genera siempre, aunque encuentre tipos no reconocidos, en tal caso, spoonEMF indica por consola los problemas que ha ido encontrando. Todas aquellas clases que instancian tipos no reconocidos no se incluyen en el metamodelo. Ver el [Apéndice B](#) de este documento para obtener información más detallada.

Una vez implementado el componente spoonClient para generar una instancia del metamodelo de Java a partir de un programa Java, se tomó la decisión de generar un plugin de Eclipse que ejecute esta funcionalidad desde el propio Eclipse. Para ello se investigó la creación de los distintos tipos de plugins desde Eclipse y posteriormente se implementó el plugin **es.ucm.spoon.client** que utiliza el sistema para generar los metamodelos de los programas Java. En el apartado [Módulo es.ucm.spoon.client](#) se detalla el diseño y la implementación de este plugin.

1.3.3 JET Java Emitter Template

Java Emitter Template (JET) [\[http://www.vogella.de/articles/EclipseJET/article.html\]](http://www.vogella.de/articles/EclipseJET/article.html) es un subproyecto de Eclipse Modeling Framework (EMF) para simplificar el proceso de generación automática de código (Java, XML, JSP, etc.) a partir de plantillas o templates.

JET funciona a partir de plantillas muy similares a los JSP (JavaServer Pages) que, al igual que en esta tecnología, son traducidas a una clase Java para, posteriormente, ser ejecutada.

En general, JET se utiliza para generar librerías que van encapsuladas dentro de un plugin con interfaz de usuario tipo asistente; o bien, se puede encapsular dentro de una tarea Ant.

JET utiliza plantillas para generar código fuente y a su vez, estas plantillas pueden utilizar parámetros que establezcan determinados aspectos del código fuente resultante, estos parámetros son ficheros con formato XML. JET utiliza el lenguaje XPath [\[http://www.w3.org/TR/xpath/\]](http://www.w3.org/TR/xpath/) para procesar los ficheros XML con sus correspondientes nodos y atributos.

En este proyecto se utiliza JET para generar todo el código fuente de la aplicación a partir de plantillas de tipo *.java.jet; y los parámetros que utilizan estas plantillas son ficheros de tipo *.ecore para procesar el metamodelo de Java; de tipo *.xmi para procesar la instancia del metamodelo de Java y de tipo *.xml para procesar las métricas que se han definido en la aplicación.

1.3.4 OCL Object Constraint Language

Object Constraint Language (OCL) es un estándar definido por Object Management Group (OMG®) [5].

Object Constraint Language [4] es un lenguaje estándar que permite describir expresiones sobre un modelo UML, estas expresiones suelen especificar las condiciones invariantes que debe

cumplir el sistema que está siendo modelado o puede especificar consultas sobre los objetos descritos en el modelo.

Al evaluar expresiones OCL sobre un sistema determinado, no se altera el comportamiento del mismo. Las expresiones OCL se pueden utilizar para especificar acciones u operaciones que no alteran el estado del sistema y se puede utilizar para definir restricciones o consultas sobre un determinado modelo. En nuestro proyecto, el sistema utiliza OCL para especificar consultas sobre la instancia del metamodelo de Java.

OCL es un lenguaje de especificación pura, y por lo tanto, se garantiza que la evaluación de una expresión OCL sea sin efectos secundarios; cuando se evalúa una expresión OCL simplemente se devuelve un valor y no cambia nada en el modelo, de hecho, la evaluación de una expresión OCL es instantánea.

1.3.5 EOS A Java component for OCL evaluation

EOS es un componente Java que permite la evaluación de expresiones OCL 2.0 [3]. EOS puede ser utilizado por otras herramientas de modelado para obtener soporte a las distintas funcionalidades de OCL, por ejemplo, validación, análisis y medición de modelos.

El API de EOS [<http://maude.sip.ucm.es/eos/v0.3/doc/index.html>] proporciona métodos para insertar por un lado los elementos del modelo en cuestión, y por otro las expresiones OCL que se desean evaluar.

El sistema utiliza el componente EOS para evaluar las métricas definidas sobre el programa Java que se desea evaluar, obteniendo la información de la instancia del metamodelo de Java.

EOS recibe tres entradas, inicialmente se inserta el diagrama de clases, que en nuestro caso se corresponde con la instancia del metamodelo de Java, posteriormente se inserta el diagrama de objetos que en nuestro caso se corresponde con el metamodelo del programa Java en cuestión y por último se van ejecutando dinámicamente las distintas consultas OCL. El apartado [Ejecución de la aplicación](#) del tercer capítulo Diseño técnico se especifica cómo se evalúan las métricas a través de esta herramienta.

2. Estado del arte

En este apartado se pretende dar una visión general del estado del arte de las métricas de calidad del software, así como de las herramientas existentes hasta el momento que permite la evaluación de este tipo de métricas sobre un determinado software.

2.1 Métricas de calidad del software

Los sistemas de métricas de calidad del software tradicionales se han centrado fundamentalmente en las métricas de procesos, de productos y de recursos [8]. Los sistemas de métricas para tiempo de ejecución más comunes hoy en día son los usados en los profilers o aplicaciones para probar las aplicaciones [7]. Este tipo de aplicaciones usan sistemas de métricas en tiempo de ejecución para medir tiempos, buscar cuellos de botella en las aplicaciones, medir capacidades máximas, etcétera. Así, las métricas tratan de servir de medio para entender, monitorizar, controlar, predecir y probar el desarrollo software y los proyectos de mantenimiento (Briand et al., 1996)

Los tres objetivos fundamentales de la medición son (Fenton y Pfleeger, 1997):

- Entender qué ocurre durante el desarrollo y el mantenimiento.
- Controlar qué es lo que ocurre en nuestros proyectos.
- Mejorar nuestros procesos y nuestros productos.

En ingeniería del software, la medición es una disciplina relativamente joven, y no existe consenso general sobre la definición exacta de los conceptos y terminología que maneja. Proporcionamos a continuación las definiciones del Institute of Electrical and Electronics Engineers (IEEE):

Definiciones generales:

Métrica: medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado (IEEE, 1993). Incluye el método de medición.

Medición: proceso por el cual se obtiene una medida.

Medida: valor asignado a un atributo de una entidad mediante una medición.

La métrica tradicionalmente más relevante de la complejidad del software es la llamada *Complejidad ciclomática*, que proporciona una medición cuantitativa de la complejidad lógica de un programa [9] como el número de caminos independientes dentro de un fragmento de código. En general los resultados se interpretan dentro de estos rangos: una complejidad ciclomática de 1 a 10 es un programa simple sin mucho riesgo; de 10 a 20 es un riesgo más complejo; de 21 a 50, muy complejo, un programa de alto riesgo y más de 50, programa no testeable (ya que la

métrica indica el número mínimo de caminos independientes que han de ejecutarse (tests) para estar seguros de haber ejecutado al menos una vez todas las sentencias de un programa y todas las condiciones lógicas en sus vertientes verdadera y falsa). Se calcula aplicando, por ejemplo, la siguiente fórmula: $M = \text{Numero de condiciones} + 1$.

Las métricas para sistemas orientados a objetos deben de ajustarse a las características que distinguen el software orientado a objetos del software convencional [6]. Estas métricas hacen hincapié en el encapsulamiento, la herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas orientadas a objetos se centran en métricas que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hagan única a una clase concreta.

Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estructurado:

- Comprender mejor la calidad del producto.
- Estimar la efectividad del proceso.
- Mejorar la calidad del trabajo realizado en el nivel del proyecto.

Berard [Laranjeira '90] define cinco características que dan lugar a unas métricas especializadas:

- **Localización:** indica la forma en que se concentra la información dentro de un programa. La información se concentra mediante el encapsulamiento tanto de datos como de procesos dentro de los límites de una clase u objeto.
- **Encapsulamiento:** Berard [Pressman '07] define el encapsulamiento como “el empaquetamiento (o enlazado) de una colección de elementos”. El encapsulamiento comprende las responsabilidades de una clase, incluyendo sus atributos (y otras clases para objetos agregados) y operaciones, y los estados de la clase, según se definen mediante valores específicos de atributos.
- **Ocultamiento de información,** un sistema orientado a objetos bien diseñado debería de impulsar al ocultamiento de información. El ocultamiento de información suprime los detalles operativos de un componente de un programa. Tan sólo se proporciona la información necesaria para acceder a ese componente o a aquellos otros componentes que deseen acceder a él.
- **Herencia:** la herencia es un mecanismo que hace posible que los compromisos de un objeto se difundan a otros objetos. La herencia se produce a lo largo de todos los niveles de la jerarquía de clases.
- **Técnicas de abstracción de objetos:** mecanismo que permite al diseñador centrarse en los detalles esenciales de algún componente de un programa.

Se sabe que la clase es la unidad principal de todo sistema orientado a objetos. Por consiguiente, las medidas y métricas para una clase individual, la jerarquía de clases, y las colaboraciones de

clases resultarán sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño. Se ha visto que la clase encapsula a las operaciones (procesamiento) y a los atributos (datos). La clase suele ser el "padre o ancestro" de las subclases (que a veces se denominan "descendientes") que heredan sus atributos de operaciones. La clase suele colaborar con otras clases.

Puesto que aquí tratamos de métricas definidas sobre diseño o métricas independientes de plataforma sobre modelos, podrían surgir problemas debidos a pérdida de información que se produzca en la traducción del código fuente a las instancias concretas [10]. En este proyecto confiamos en la traducción realizada por la herramienta Spoon.

Basándose en lo definido por [Archer y Stinson, 95], [García y Harrison, 2000] se describen un conjunto representativo de métricas orientadas a objetos [11], cuyo correlación con número de defectos o esfuerzos de mantenimiento ya ha sido validada [Abreu y Melo, 1996] y [Basili et Al., 1995]. Contamos así con:

- Métricas a Nivel de Sistema.
- Métricas de Acoplamiento.
- Métricas de Herencia.
- Métricas de Clases.
- Métricas de métodos.

En estos grupos se establecen determinados rangos de valores fuera de los cuales una clase o pieza del código es más propensa a errores.

A continuación se presentan distintos grupos de métricas que por ser los más conocidos, representativos o utilizados, son los que se han considerado para este trabajo.

2.1.1 Métricas CK Chidamber y Kemerer

Son métricas orientadas a clases: clases individuales, herencia y colaboraciones. Es uno de los conjuntos de métricas más referenciado.

- a) **Métodos ponderados por clase (WMC: Weighted Methods per Class).** Calcula la suma de la complejidad ciclomática de los métodos de una clase:

$$WMC = \sum_{i=1..n} mc_i, \text{ siendo } mc_i \text{ la complejidad ciclomática del método } i.$$

El WMC debe ser lo más bajo posible. Cuanto más alto es el valor WMC, más complejo el árbol de herencia y menos reutilizable.

Las principales interpretaciones de esta métrica son las siguientes:

- El número y la complejidad de los métodos son indicadores del tiempo necesario para desarrollar/mantener la clase.
- Cuanto mayor sea el nº de métodos mayor impacto potencial tendrá en los hijos, sus herederos potenciales.

- Las clases con gran nº de métodos serán de aplicación específica, y por lo tanto más difíciles de reutilizar.

b) **Profundidad en el árbol de herencia (DIT: Depth Inheritance Tree).** Es la distancia desde una clase a la raíz del árbol de herencia. Cuanto más alto es el mayor valor de DIT, mayor complejidad hay en el diseño y cuanto más alto sea el valor de DIT de una clase más posibilidades existen de que reutilice/refine métodos heredados.

Las principales interpretaciones de esta métrica son las siguientes:

- A mayor profundidad de la clase, más métodos puede heredar y es más difícil de explicar su comportamiento.
- A mayor profundidad de una clase, mayor posibilidad de reutilización de métodos heredados.

c) **Número de hijos inmediatos en el árbol de herencia (NOC: Number Of Children).** En principio, cuanto más alto es el valor de NOC, una clase es más reutilizable pero también la probabilidad de que se hayan hecho extensiones no apropiadas de la clase es mayor.

Las principales interpretaciones de esta métrica son las siguientes:

- Cuanto mayor sea NOC más reutilización habrá por herencia.
- Si NOC es muy grande hay un fallo en la abstracción de la clase padre, falta algún nivel intermedio.
- NOC da una idea del peso que la clase tiene en el diseño, y de los recursos que se deben dedicar a probar sus métodos.

d) **Acoplamiento entre clases (CBO: Coupling Between Object Classes).** Es el número de clases acopladas a una clase. Dos clases están acopladas cuando los métodos de una de ellas usan variables o métodos de una instancia de la otra clase. Si existen varias dependencias sobre una misma clase es computada como una sola. No es deseable que $CBO > 14$. Cuanto más alto es el más difícil será el mantenimiento y el reuso y en general el código será más propenso a fallos.

Las interpretaciones de esta métrica son las siguientes:

- Cuanto mayor es CBO, peor es la modularidad y la reutilización.
- Cuanto mayor es CBO, peor es el encapsulamiento y más cuesta mantenerlo.

e) **Respuesta para una clase (RPC: Response for a Class).** Es el número de métodos que pueden ser ejecutados en respuesta a un mensaje recibido por un objeto de esa clase. Cuanto mayor sea RPC, mayor esfuerzo se requiere para su comprobación, y más complejo es el diseño. Existen dos variaciones de esta métrica:

- I. **RPC:** Sólo considera el primer nivel del árbol de llamadas: número de métodos de una clase más el número de métodos remotos llamados directamente por una clase.

- II. **RPC'**: Considera todo el árbol de llamadas: número de métodos de una clase más el número de métodos remotos llamados recursivamente por una clase considerando el árbol entero de llamadas.

Las principales interpretaciones de esta métrica son las siguientes:

- a. Cuanto mayor sea la respuesta de una clase, más complicadas serán las pruebas y el mantenimiento.
- b. Cuanto mayor sea la respuesta de una clase, mayor será la complejidad de la clase.

- f) **Carencia de cohesión (LCOM: Lack of Cohesion)**. Cada método de una clase tiene acceso a uno o más atributos. LCOM calcula el conjunto de atributos comunes en los métodos de una clase. Dos métodos son similares si comparten al menos un atributo de la clase. A mayor número de atributos similares, mayor cohesión hay en la clase.

Ejemplos:

- III. Si ningún método accede a sus atributos, LCOM=0.
- IV. Una clase tiene 6 métodos y 4 de ellos tienen un atributo en común, LCOM=4

Es interesante conseguir valores bajos de LCOM. Las principales de esta métrica son las siguientes:

- a. A mayor cohesión mayor encapsulamiento.
- b. Un valor grande puede indicar que la clase debe dividirse.
- c. Baja cohesión indica alta complejidad y alta probabilidad de error en el desarrollo.

2.1.2 Métricas de Lorenz y Kidd (1994)

En el libro de métricas realizado por Lorenz y Kidd [Laranjeira '90], dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos.

Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema orientado a objetos en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases.

Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

- a) Métricas de **tamaño**:

- I. **PIM**: Número de métodos de instancia públicos.
- II. **NIM**: Todos los métodos de una instancia.
- III. **NIV**: Todas las variables de una instancia.
- IV. **NCM**: Todos los métodos de una clase.
- V. **NIM**: Todas las variables de una clase.

- b) Métricas de **herencia**:

- I. **NMO**: Número de métodos sobrecargados.
- II. **NMI**: Número de métodos heredados.

- III. **NMA**: Número de métodos añadidos, número total de métodos que se definen en una subclase.
- IV. **SIX**: Índice de especialización para cada clase. Cómo una subclase redefine el comportamiento de superclase.
Fórmula: (Número de métodos sobrescritos * nivel de anidamiento jerarquía) / número total de métodos
- c) Métricas de **características internas de las clases**.
 - I. **APPM** Promedio de parámetros por método
Número total de parámetros por método / Número total de métodos.

2.1.3 Métricas R. Martin (1994)

- a) **Ca: Afferent Couplings**: número de clases de otros paquetes que dependen de las clases del propio paquete.
- b) **Ce: Efferent Couplings**: número de clases dentro del propio paquete que dependen de clases de otros paquetes.
- c) **I: Instability** = $Ce / (Ca + Ce)$, métrica comprendida entre [0,1], siendo 0 la máxima estabilidad y 1 máxima inestabilidad.
- d) Métrica que mide la **abstracción** de un paquete:
 - I. **NAC** número de clases abstracta en un paquete.
 - II. **NTC** número total de clases en un paquete.
 - III. Esta métrica está comprendida entre [0,1], siendo 0 completamente concreto y 1 completamente abstracto.

2.1.4 Métricas de Li-Henry

- a) **APM** Acoplamiento por paso de mensajes: número de mensajes o métodos invocados enviados por una clase a otras clases del sistema.
- b) **AAD** Acoplamiento por abstracción de datos: (se considera una definición ambigua con dos posibles interpretaciones):
 - I. Número de tipos de datos abstractos definidos en una clase.
 - II. Número de atributos de una clase que se referencian desde otra clase.
- c) **NML** Número de métodos localmente definidos en la clase.
- d) **Tamaño**: número de atributos y métodos locales de una clase.

2.2 Herramientas de evaluación de métricas para programas Java.

En este apartado se muestran algunas de las herramientas ya existentes que comprueban código fuente de aplicaciones Java validando su estructura y evaluando la calidad del software.

2.2.1 JCSC – Java Coding Standard Checker

JCSC [<http://jcsc.sourceforge.net/>] es una herramienta que comprueba el código fuente contra estándares de codificación muy definibles, el estándar cubre convenciones de nomenclatura para las clases, interfaces, atributos, parámetros, etcétera; además se puede definir la arquitectura de clases, como por ejemplo, la ubicación y orden de los atributos, si es

antes o después de los métodos. Otra de las funcionalidades de esta herramienta es buscar debilidades en el código, como por ejemplo: capturas de excepciones vacías, switch sin la opción default, etcétera.

Para la verificación del código fuente, la herramienta permite al usuario definir una serie de reglas que deberá tener en cuenta a la hora de comprobar el código fuente, es decir, comprueba que se cumplen todas las reglas especificadas por el usuario. JCSC cuenta con una interfaz gráfica de usuario para la configuración de reglas:

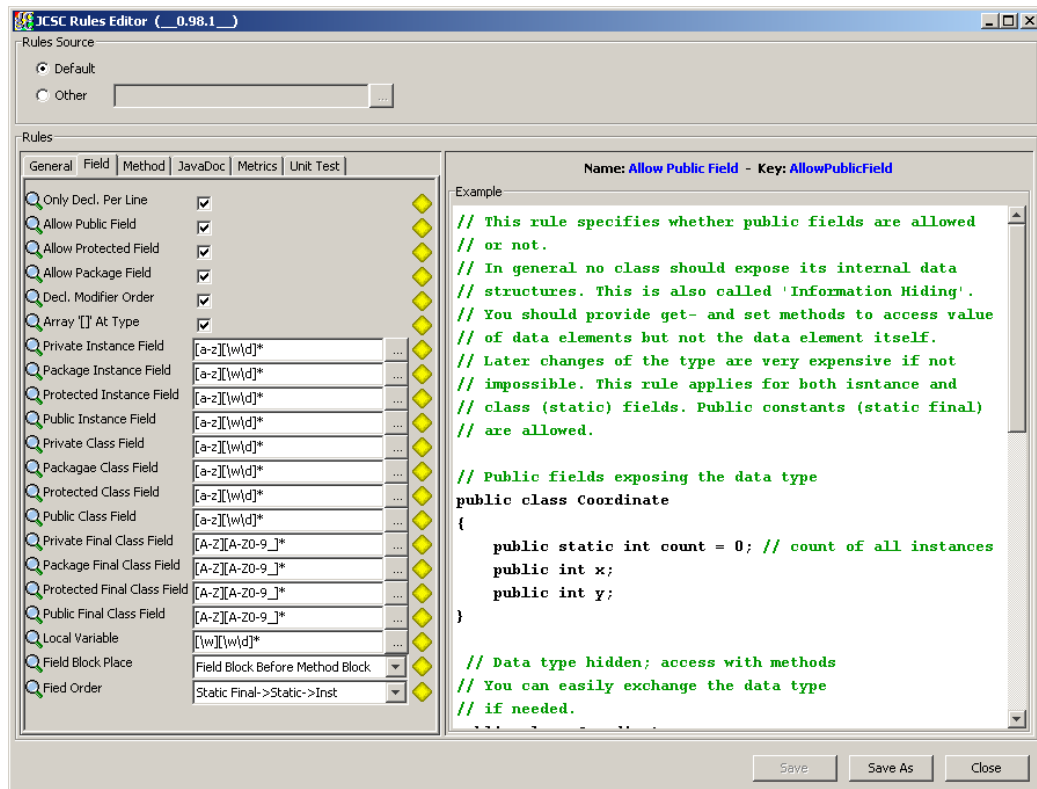


Ilustración 2: Estado del arte: Herramienta JCSC.

Además de analizar sintácticamente el código fuente, la herramienta JCSC evalúa la calidad del código a partir de las siguientes métricas:

- **NCSS Non Commenting Source Statements:** representa el número de líneas de código útiles sin incluir los comentarios.
- **CNN Cyclomatic Complexity Number:** representa el número de caminos que se pueden ejecutar en cada método o constructora, por defecto cada método tiene un CNN de 1.

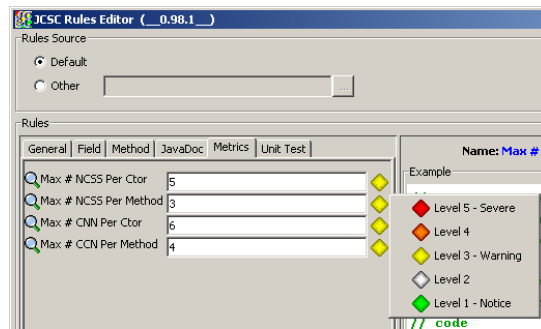


Ilustración 3: Estado del arte: Herramienta JCSC – Configuración de métricas.

Como conclusiones, de esta herramienta podemos decir que es una herramienta sencilla, que a nivel de estructura del código fuente permite definir numerosas reglas para conseguir un código fuente elegante. Sin embargo, para evaluar la calidad del código tan sólo define dos métricas.

2.2.2 CheckStyle

CheckStyle [<http://checkstyle.sourceforge.net/>] es una herramienta de desarrollo para ayudar a los programadores a escribir código Java que se adhiere a un estándar de codificación. Al igual que la herramienta anterior, comprueba el código fuente para que cumpla determinadas reglas de codificación, indicando aquellas que no cumple y el grado de severidad que se ha considerado en cada caso.

CheckStyle permite al usuario especificar en un fichero XML las siguientes métricas para la evaluación de la calidad del software:

- **BooleanExpressionComplexity:** especifica el número máximo de operadores (&&, ||, &, |,...) en una expresión condicional.
- **ClassDataAbstractionCoupling:** especifica el número de instancias a otras clases dentro de una determinada clase.
- **ClassFanOutComplexity:** especifica el número de veces que una determinada clase es instanciada desde otras clases del programa.
- **CyclomaticComplexity:** especifica el límite de complejidad ciclomática. La complejidad ciclomática se mide por el número de expresiones if, while, do, for, ?, catch, switch, case y operaciones lógicas &&, || en el cuerpo de constructoras o métodos. Sus valores se interpretan considerando que de 1 a 4 es muy bueno, de 5 a 7 es bueno, de 8 a 10 se debería considerar la refactorización el código y superior a 11 se debe refactorizar.
- **NPathComplexity:** especifica el número de posibles caminos de ejecuciones dentro de una determinada función.
- **JavaNCSS:** determina la complejidad de los métodos, clases y archivos contando las Non Commenting Source Statements (NCSS). Esta comprobación se adhiere a la especificación de la herramienta [JavaNCSS](#) escrita por Chr. Clemens Lee. La métrica NCSS se calcula contando el número de las líneas que no son comentarios. El

NCSS para una clase es un resumen del NCSS de todos sus métodos, el NCSS de sus clases anidadas y el número de declaraciones de variables locales. Los métodos demasiado grandes y las clases son difíciles de leer y costosas de mantener. Un gran número de NCSS a menudo significa que un método o una clase tiene demasiadas responsabilidades y / o funcionalidades que deben ser descompuestas en unidades más pequeñas.

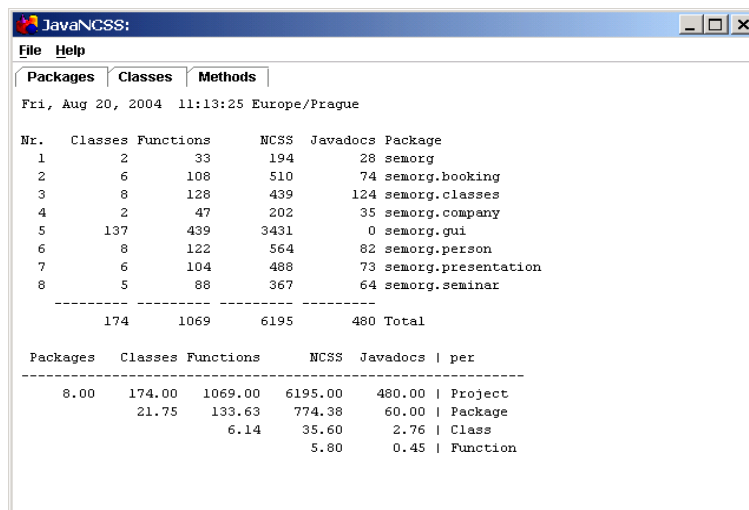
Como conclusiones de esta herramienta podemos decir que es una herramienta útil para reportar posibles errores, es decir, para reportar aquellos puntos del código que no cumplen las reglas especificadas por el usuario, pero dispone de un número reducido de métricas.

2.2.3 JavaNCSS

JavaNCSS [<http://javancss.codehaus.org/>] es una herramienta que nos permite realizar mediciones sobre el código fuente Java, obteniendo los valores de dichas mediciones agrupados a nivel global, de clase y a nivel de función.

Entre las métricas obtenidas por esta herramienta destacan las siguientes:

- **Número de clases por paquete.** El número de clases por paquete nos dan una idea de su tamaño y responsabilidad, así como de la cantidad de funcionalidad cubierta.
- **Número de métodos por paquete.** Al igual que la métrica anterior, analiza la responsabilidad del paquete, es decir, la cantidad de funcionalidad que cubre.
- **Número de líneas de código exceptuando comentarios.** Esta métrica proporciona un valor (NCSS) de la cantidad de código que contiene cada clase, paquete y método.
- **Número de bloques de documentación Javadoc.**
- **Número de líneas de comentario.**
- **Complejidad Ciclomática.** Es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. La métrica, propuesta por Thomas McCabe en 1976, se basa en la representación gráfica del flujo de control del programa y se calcula en función del número de puntos de decisión del programa.

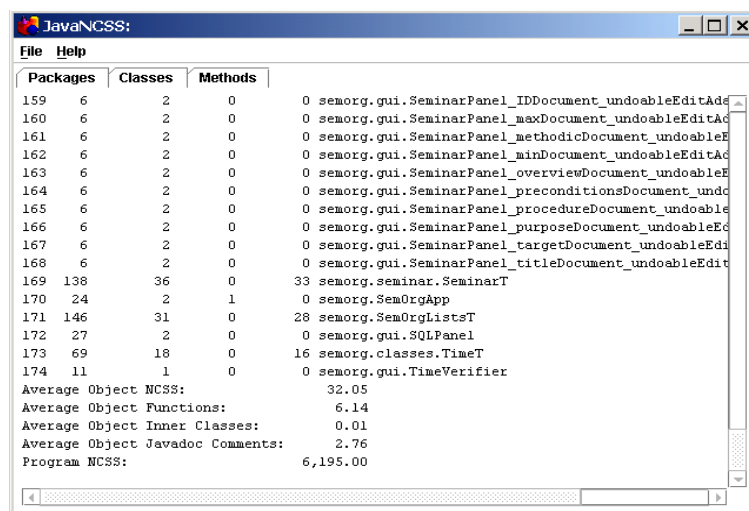


Nr.	Classes	Functions	NCSS	Javadocs	Package
1	2	33	194	28	semorg
2	6	108	510	74	semorg.booking
3	8	128	439	124	semorg.classes
4	2	47	202	35	semorg.company
5	137	439	3431	0	semorg.gui
6	8	122	564	82	semorg.person
7	6	104	488	73	semorg.presentation
8	5	88	367	64	semorg.seminar

	174	1069	6195	480	Total

Packages	Classes	Functions	NCSS	Javadocs	per
8.00	174.00	1069.00	6195.00	480.00	Project
	21.75	133.63	774.38	60.00	Package
		6.14	35.60	2.76	Class
			5.80	0.45	Function

Ilustración 4: Estado del arte: Herramienta JavaNCSS – Métricas por paquete.



Nr.	Classes	Functions	NCSS	Javadocs	Package
159	6	2	0	0	semorg.gui.SeminarPanel_IDocument_undoableEditAd
160	6	2	0	0	semorg.gui.SeminarPanel_maxDocument_undoableEditAd
161	6	2	0	0	semorg.gui.SeminarPanel_methodicDocument_undoableE
162	6	2	0	0	semorg.gui.SeminarPanel_minDocument_undoableEditAd
163	6	2	0	0	semorg.gui.SeminarPanel_overviewDocument_undoableE
164	6	2	0	0	semorg.gui.SeminarPanel_preconditionsDocument_undoc
165	6	2	0	0	semorg.gui.SeminarPanel_procedureDocument_undoable
166	6	2	0	0	semorg.gui.SeminarPanel_purposeDocument_undoableEd
167	6	2	0	0	semorg.gui.SeminarPanel_targetDocument_undoableEdi
168	6	2	0	0	semorg.gui.SeminarPanel_titleDocument_undoableEdit
169	138	36	0	33	semorg.seminar.SeminarT
170	24	2	1	0	semorg.SemOrgApp
171	146	31	0	28	semorg.SemOrgListsT
172	27	2	0	0	semorg.gui.SQLPanel
173	69	18	0	16	semorg.classes.TimeT
174	11	1	0	0	semorg.gui.TimeVerifier

Packages	Classes	Functions	NCSS	Javadocs	per
Average Object NCSS:			32.05		
Average Object Functions:			6.14		
Average Object Inner Classes:			0.01		
Average Object Javadoc Comments:			2.76		
Program NCSS:			6,195.00		

Ilustración 5: Estado del arte: Herramienta JavaNCSS – Métricas por clase.

JavaNCSS:

File Help

Packages	Classes	Methods
1054	9	2
1055	2	1
1056	2	1
1057	2	1
1058	2	1
1059	2	1
1060	2	1
1061	2	1
1062	2	1
1063	2	1
1064	2	1
1065	2	1
1066	2	1
1067	2	1
1068	4	1
1069	20	6
1070	10	6
Average Function NCSS:		4.43
Average Function CCN:		1.40
Average Function JVDC:		0.41
Program NCSS:		6,195.00

Ilustración 6: Estado del arte: Herramienta JavaNCSS – Métricas por método.

Como conclusiones de esta herramienta podemos decir que es una herramienta más robusta que las dos anteriores, ofrece una interfaz de usuario sencilla y fácil de usar pero el número de métricas que ofrece es muy limitado, sin embargo si ofrece una evaluación más concreta del código, puesto que evalúa paquetes, clases y métodos concretos.

2.2.4 JMT

JMT Java Measurement Tool [<http://www-ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/jmt.html>] de Christian Kolbe es una herramienta que analiza las clases Java y las relaciones entre ellas.

Existen dos posibles formas de analizar las clases Java: permite analizar una clase Java concreta con la opción *“Analyze a file”* o permite analizar un proyecto Java completo con la opción *“Analyze a Project”*. Para analizar un proyecto Java completo es necesario cargar individualmente cada una de las clases Java con la opción *“Load a file”*.

Las métricas que utiliza la herramienta JMT para medir la calidad del software son las siguientes:

- Métricas por clase:
 - **DIT** Depth of Inheritance Tree: especifica la máxima longitud del camino entre la clase y la clase padre o raíz.
 - **NOC** Number of Children: especifica el número directo de sucesores de la clase.
 - **WMC** Weighted Methods per Class: especifica el número de métodos de la clase en cuestión.
 - **WAC** Weighted Attributes per Class: especifica el número de atributos de la clase en cuestión.
 - **CBO** Coupling between Object Classes: especifica el número de clases que referencian algún método o atributo de la clase en cuestión.
 - **PIM** Number of Public Methods.
 - **NMI** Number of Methods inherited: especifica el número de métodos heredados de las clases de las que hereda directamente.
 - **NAI** Number of Attributes inherited: especifica el número de atributos heredados de las clases de las que hereda directamente.
 - **NMO** Number of Methods overwritten.
 - **RFC** Response for a class: especifica el número de métodos de la propia clase más el número de métodos externos que utiliza la clase en cuestión.
 - **LOC** Lines of Code.
- Métricas por método:

- **NOP** Number of Parameter.
- **LOC** Lines of Code.
- Métricas de herencia:
 - **MIF** Method Inheritance Factor: especifica la relación entre el número de métodos heredados y el número total de métodos.
 - **AIF** Attribute Inheritance Factor: especifica la relación entre el número de atributos heredados y el número total de atributos.
- Métricas generales del sistema:
 - **COF** Coupling Factor: ejecuta la siguiente fórmula:

$$(\text{Total number of couplings}) / (n^2 - n); \text{ siendo } n \text{ el número de clases definidas en el sistema.}$$
 - **ANM** Average Number of Methods per Class.
 - **ANM** Average Number of Attributes per Class.
 - **ANM** Average Number of Parameter per Method.

La siguiente ilustración muestra un ejemplo de ejecución de la herramienta JavaNCSS:

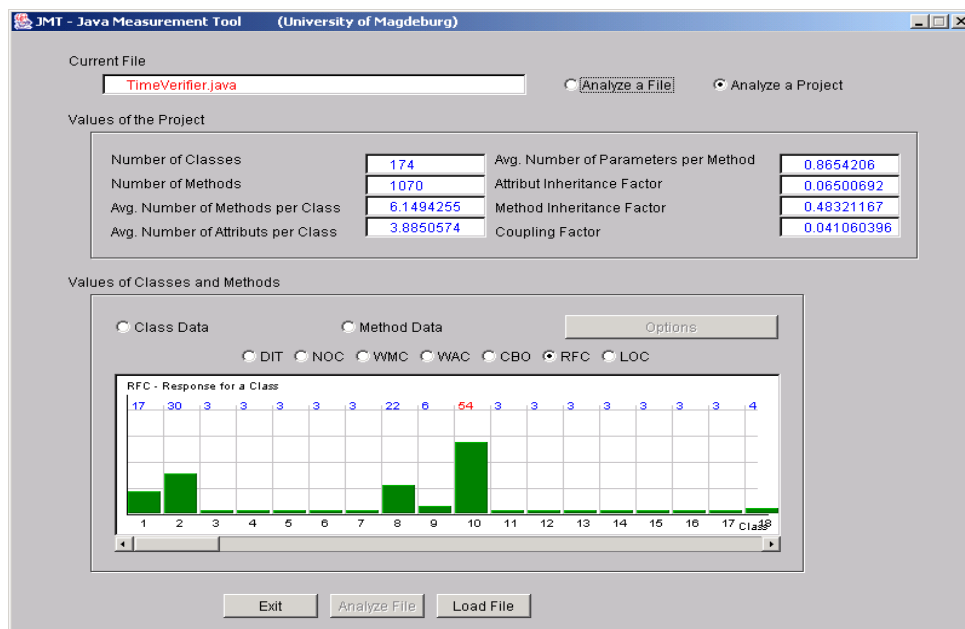


Ilustración 7: Estado del arte: Herramienta JMT.

Como conclusiones de esta herramienta podemos decir que es la herramienta más completa entre las herramientas descritas hasta el momento, puesto que permite evaluar un mayor número de métricas que las anteriores. Sin embargo, tiene los siguientes inconvenientes: aunque permite evaluar todo el proyecto conjuntamente, el usuario deberá cargar cada una de

las clases individualmente, ejecutando para cada una de ellas diferentes pasos (Analyse a File → Load File → Select File → Open File → Analyse File → OK), lo cual impacta para mal la usabilidad de la herramienta, impidiendo incluso que pueda escalarse su uso a grandes proyectos.

2.2.5 Metrics Eclipse plugin

Metrics 1.3.6 [<http://metrics.sourceforge.net/>] es un plugin de Eclipse que define un conjunto de métricas permitiendo al usuario especificar el mínimo y el máximo resultado permitido para cada una de ellas.

La siguiente ilustración muestra las métricas definidas por este plugin:

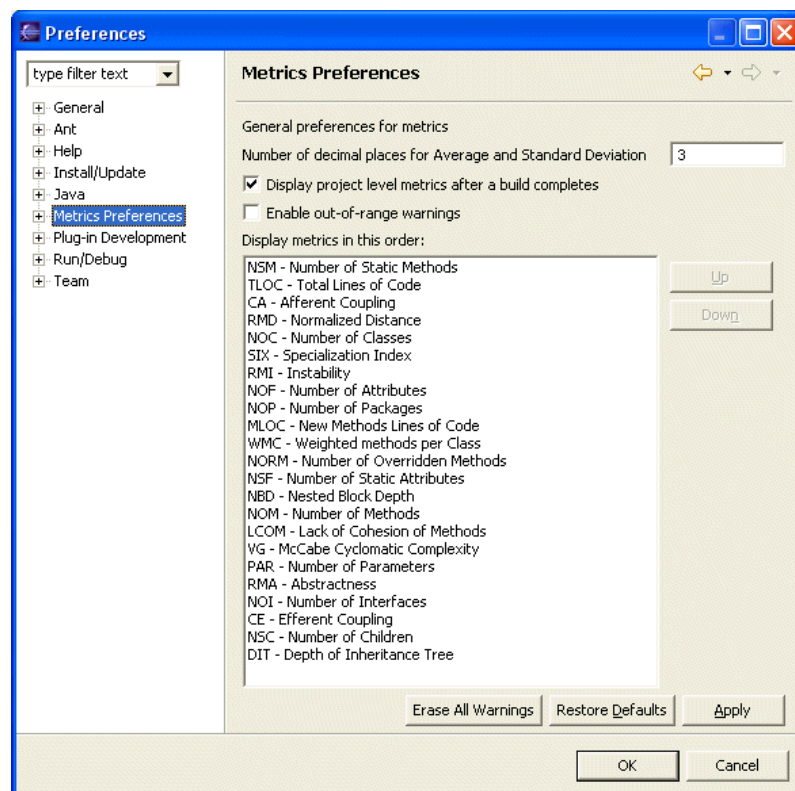


Ilustración 8: Estado del arte: Eclipse plugin Metrics 1.3.6 - Preferencias

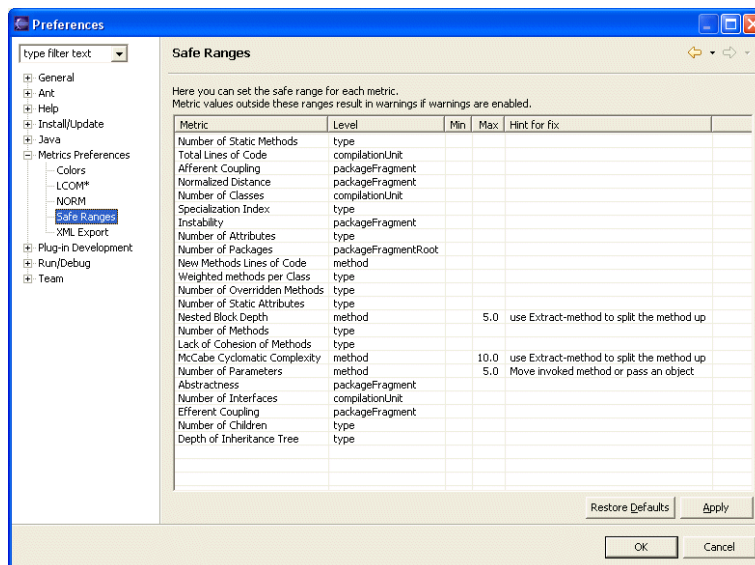


Ilustración 9: Estado del arte: Eclipse plugin Metrics 1.3.6 - Configuración

Evalúa un determinado proyecto Java indicando los resultados que están dentro del rango especificado por el usuario en azul y los resultados que están fuera del rango en rojo.

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Packages	16					
Number of Methods (avg/max per type)	1310	6.65	8.553	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
tgsrc	489	7.191	11.544	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
src	761	6.238	6.553	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core.sources	108	15.429	12.129	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui	77	9.625	10.111	33	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core	199	6.6	7.093	27	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.preferences	52	6.5	7.467	26	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.dependencies	95	5.588	3.727	15	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.persistence	18	4.5	4.33	12	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler.implementa...	54	5.4	2.871	10	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.xml	41	4.1	2.022	9	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.calculators	79	4.158	2.254	8	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.propagators	31	5.167	1.067	7	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.tests	8	2.667	1.886	4	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler	0	0	0	0		
classycle	60	8.571	2.556	13	/net.sourceforge.metrics/classycle/classycle/g...	
Lines of Code (avg/max per type)	6593	33.467	49.02	339	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
Number of Interfaces (avg/max per packageFragment)	16	1	1.414	4	/net.sourceforge.metrics/src/net/sourceforge/...	
Lines of Code (avg/max per method)	6593	4.812	7.355	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
classycle	324	5.4	9.94	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
tgsrc	2321	4.661	8.278	59	/net.sourceforge.metrics/tgsrc/com/touchgrap...	scrollSelectPanel
src	3948	4.862	6.473	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
net.sourceforge.metrics.ui	544	6.8	8.707	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable.java	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
setMetrics	52					

Ilustración 10: Estado del arte: Eclipse plugin Metrics 1.3.6 - Resultados

Como funcionalidades destacables de este plugin, están la opción de generar un diagrama de dependencias y la posibilidad de exportar en XML los resultados obtenidos.

Como conclusiones, se puede destacar que es una herramienta muy completa y fácilmente usable que muestra al usuario los resultados de evaluación de proyectos Java en el mismo entorno de trabajo. Como inconveniente, no permite al usuario añadir más métricas de evaluación.

2.2.6 RSM Resource Standard Metrics

RSM Resource Standard Metrics [<http://msquaredtechnologies.com/m2rsm/>] es una herramienta de análisis de calidad que proporciona un método estándar para el análisis de C, ANSI C ++, C # y el código fuente de Java en sistemas operativos.

Esta herramienta tiene definidas numerosas métricas agrupadas en métricas sobre funciones, métricas sobre clases, métricas sobre paquetes y métricas generales del proyecto. La definición de estas métricas puede leerse a través del siguiente enlace:

http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics.htm#File%20metrics

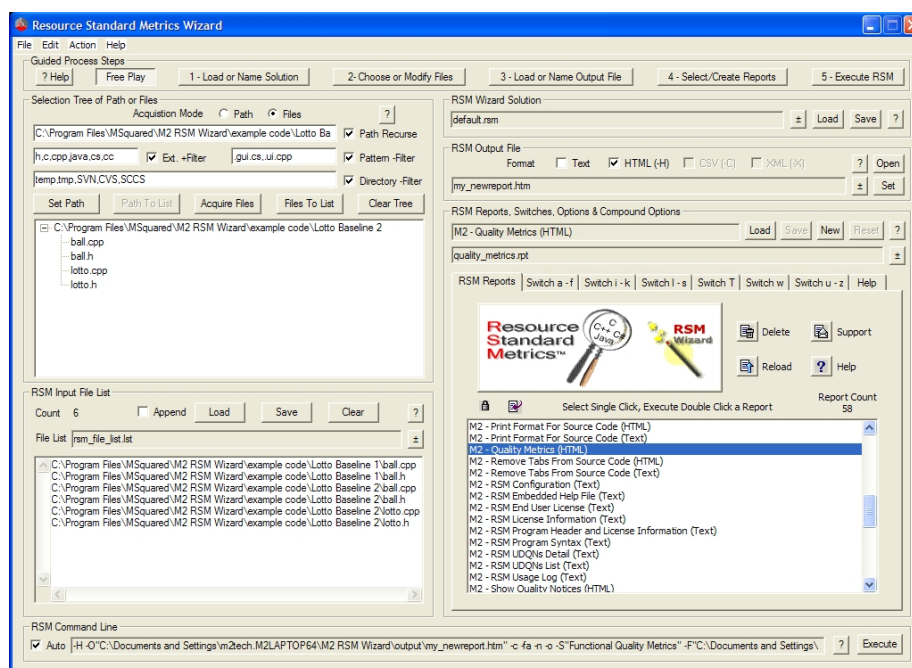


Ilustración 11: Estado del arte: Herramienta RSM

RSM es una herramienta basada en comandos, es decir se puede ejecutar desde una consola UNIX. Esto permite que RSM para integrarse sin problemas en Visual Studio, Kawa y otros IDE, como por ejemplo Eclipse. RSM utiliza un archivo de licencia y archivo de configuración en el arranque. El archivo de configuración permite al usuario personalizar el grado de análisis de código fuente. RSM típicamente toma conmutadores de tiempo de ejecución que permite al usuario personalizar la salida deseada.

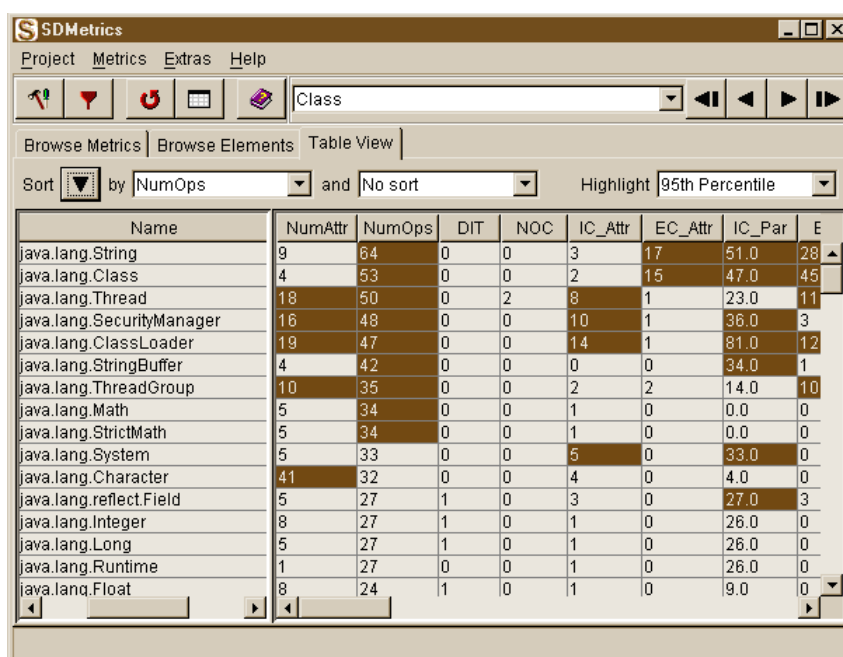
RSM es la herramienta más completa de las que hemos visto hasta el momento, destacan los distintos lenguajes de programación que es capaz de evaluar, todas las opciones de configuración y personalización que ofrece al usuario, su capacidad de integrarse con otros IDE. Esta herramienta requiere de licencia para poder utilizarse.

2.2.7 SDMetrics

SDMetrics The Software Design Metrics tool for the UML [<http://www.sdmetrics.com/index.html>] es una herramienta que analiza las propiedades estructurales de los modelos UML, para ello utiliza métricas orientadas a objetos para medir el diseño, la complejidad y la relación entre clases.

SDMetrics evalúa diagramas UML y para ello define un gran número de métricas que agrupa en: métricas de clases, métricas de interfaces, métricas de paquetes, métricas de casos de uso, métricas de máquina de estados, métricas de actividad, métricas de componentes y métricas generales de diagramas [<http://www.sdmetrics.com/LoM.html>].

La siguiente ilustración muestra un ejemplo de ejecución de esta herramienta:



The screenshot shows the SDMetrics application window. It has a menu bar (Project, Metrics, Extras, Help) and a toolbar with icons for file operations and a search icon. Below the toolbar is a 'Class' dropdown menu. The main area is divided into three tabs: 'Browse Metrics', 'Browse Elements', and 'Table View'. The 'Table View' tab is active, displaying a table of metrics. The table has columns for 'Name', 'NumAttr', 'NumOps', 'DIT', 'NOC', 'IC_Attr', 'EC_Attr', 'IC_Par', and 'E'. The table is sorted by 'NumOps' in descending order. The 'Highlight' dropdown is set to '95th Percentile'. The table lists various Java classes and their corresponding metric values.

Name	NumAttr	NumOps	DIT	NOC	IC_Attr	EC_Attr	IC_Par	E
java.lang.String	9	64	0	0	3	17	51.0	28
java.lang.Class	4	53	0	0	2	15	47.0	45
java.lang.Thread	18	50	0	2	8	1	23.0	11
java.lang.SecurityManager	16	48	0	0	10	1	36.0	3
java.lang.ClassLoader	19	47	0	0	14	1	81.0	12
java.lang.StringBuffer	4	42	0	0	0	0	34.0	1
java.lang.ThreadGroup	10	35	0	0	2	2	14.0	10
java.lang.Math	5	34	0	0	1	0	0.0	0
java.lang.StrictMath	5	34	0	0	1	0	0.0	0
java.lang.System	5	33	0	0	5	0	33.0	0
java.lang.Character	41	32	0	0	4	0	4.0	0
java.lang.reflect.Field	5	27	1	0	3	0	27.0	3
java.lang.Integer	8	27	1	0	1	0	26.0	0
java.lang.Long	5	27	1	0	1	0	26.0	0
java.lang.Runtime	1	27	0	0	1	0	26.0	0
java.lang.Float	8	24	1	0	1	0	9.0	0

Ilustración 12: Estado del arte: Herramienta SDMetrics.

SDMetrics es una herramienta que también necesita una licencia para poder utilizarla, es una herramienta muy completa con una interfaz de usuario fácilmente usable que ofrece una información muy detallada del código que se está evaluando.

2.2.8 SONAR

Sonar [<http://www.sonarsource.org/>] es una plataforma de código abierto que sirve para gestionar la calidad del código. Como tal, abarca los siguientes ámbitos de la calidad del software:

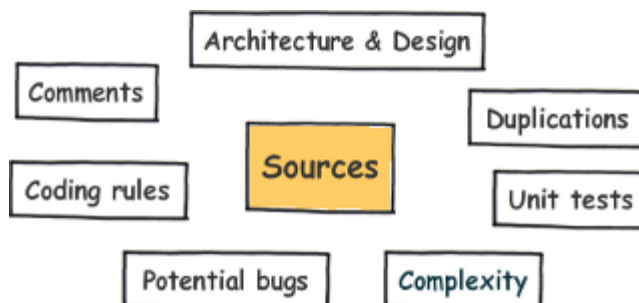


Ilustración 13: Estado del arte: Herramienta SONAR – Ámbitos de evaluación.

Sonar es una aplicación web con las siguientes características: está basada en reglas, alertas, rangos, exclusiones y configuración; permite configuración online, dispone de una base de datos y permite combinar métricas en conjunto.

Sonar permite extensiones a través de plugins para abarcar nuevos lenguajes de programación, para añadir nuevas reglas o para añadir nuevas métricas.

Inicialmente, cubre el lenguaje de programación Java, sin embargo, ya existen plugins de código libre y comerciales que extienden la herramienta para cubrir lenguajes como Flex, PL/SQL o Visual Basic.

La siguiente ilustración muestra la web de SONAR con una lista de todos los proyectos que se han ido incluyendo para llevar un seguimiento de la evaluación de cada uno de ellos.

Name	Lines of code	Technical Debt ratio	Coverage	Duplicated lines (%)	Build time	Icono
MasterProject	6.142.758	16,9%	19,0%	6,4%	2/06/2010	[Icono]
AxisLib application framework	12.167	9,7%	38,6%	1,1%	29/05/2010	[Icono]
Tapestry 5 Project	59.661	6,8%	51,2%	0,2%	31/05/2010	[Icono]
Commons Collections	20.901	10,1%	79,7%	3,3%	30/05/2010	[Icono]
MicroEmulator	28.344	11,8%		2,3%	30/05/2010	[Icono]
Apache Jackrabbit	208.717	16,7%	26,4%	4,6%	30/05/2010	[Icono]
Unitils	19.090	14,9%	2,7%	3,8%	31/05/2010	[Icono]
javagut	6.127	11,1%	61,2%	0,3%	30/05/2010	[Icono]
Wicket Parent	104.895	9,8%	36,9%	1,1%	31/05/2010	[Icono]
Commons Chain	3.901	14,4%	64,5%	21,9%	30/05/2010	[Icono]
Commons IO	6.827	5,5%	82,3%	3,1%	30/05/2010	[Icono]
Commons SCXML	7.331	9,9%	69,8%	7,2%	30/05/2010	[Icono]
Sonar	36.198	6,9%	65,6%	0,0%	2/06/2010	[Icono]
Apache Velocity	29.094	11,1%		5,1%	31/05/2010	[Icono]
Apache Abdera	52.670	8,8%		2,9%	29/05/2010	[Icono]
Castor	112.001	9,1%		6,8%	16/05/2010	[Icono]
Commons BeanUtils	11.374	15,0%	62,8%	6,6%	30/05/2010	[Icono]
JEuclyid	12.664	4,1%		0,5%	30/05/2010	[Icono]
Jackcess	11.588	5,1%	86,5%	0,2%	30/05/2010	[Icono]
jcrom	3.958	9,5%	76,5%	2,9%	30/05/2010	[Icono]

Ilustración 14: Estado del arte: Herramienta SONAR – Lista proyectos.

La siguiente ilustración muestra la web de SONAR con la evaluación concreta del proyecto CheckStyle, en el cual se puede comprobar que no hay alertas.

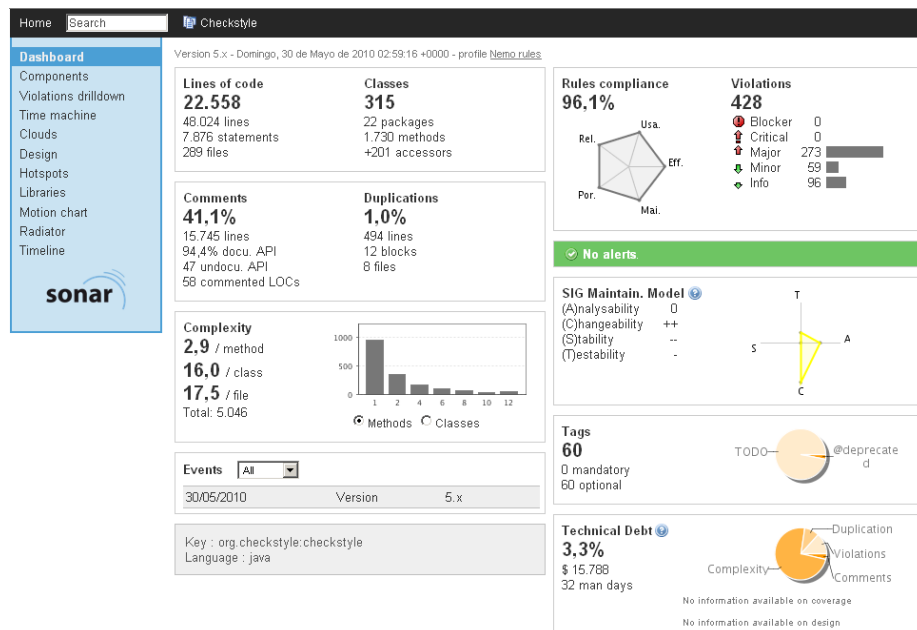


Ilustración 15: Estado del arte: Herramienta SONAR – Evaluación CheckStyle.

La siguiente ilustración muestra la web de SONAR con la evaluación concreta del proyecto JFreeChart, en el cual se puede comprobar que hay alertas por duplicación de código.

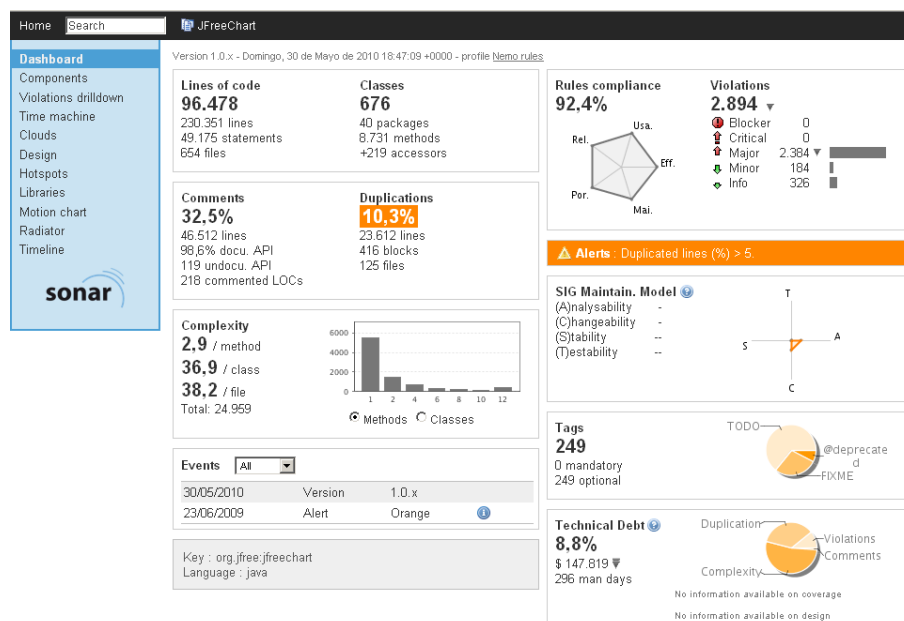


Ilustración 16: Estado del arte: Herramienta SONAR – Evaluación JFreeChart.

Las métricas que utiliza SONAR son básicamente métricas relacionadas con la complejidad ciclomática, líneas de código, código duplicado y comentarios.

2.2.9 Kemis “Kybele Environment Meseasurement Information System”

KEMIS es un proyecto llevado a cabo por Kybele Consulting S.L., Kybele Research, Dpto. de Lenguajes II de la Universidad Rey Juan Carlos, Grupo ALARCOS del Dpto. de Tecnologías y Sistemas de Información de la Universidad de Castilla-La Mancha.

Establecer sistemas de medición es una pieza básica del control de la calidad del software, más aún dentro de la actual tendencia a externalizar (outsourcing) el desarrollo, que frecuentemente es realizado por equipos o fábricas de software externos. Además, para que un sistema de medición de la calidad del producto software sea eficiente debe tener un alto nivel de automatización y posibilitar su uso de manera frecuente sin un excesivo consumo de tiempo [12].

Las categorías y métricas que se han obtenido mediante plugins de medición en el proyecto KEMIS son las siguientes:

- Código innecesario: método vacío, atributo privado no utilizado, variable local no utilizada, método privado no utilizado, parámetro no utilizado, modificador no utilizado, sentencia condicional vacía, bucle "while" vacío.
- Documentación: Javadoc en métodos, estilos Javadoc, Javadoc en clases e interfaces, estándares de documentación.
- Estilo Programación: evitar asignaciones en operandos, llamar a "super" en un constructor, bucles "for" que deberían ser "while", sentencias "if" comprimibles, evitar asignaciones en parámetros, densidad en switch, evitar inicializadores estáticos, etiqueta default no está al final de sentencia switch, convenios para nombrar clases.
- Mantenibilidad: número elevado de imports, evitar un elevado número de "if" anidados, evitar lanzar ciertos tipos de error, evitar lanzar excepciones de tipo NullPointerException, evitar literales duplicados, evitar métodos muy largos, evitar demasiados parámetros, evitar clases muy largas, complejidad ciclomática.
- Posibles errores: asignación del valor null, bloques "catch" vacíos, variable incremental entrelazada.
- Rendimiento: atributo final podría declararse como static, llamada a método toArray optimizable, evitar concatenar cadenas en un stringbuffer, inicialización explícita.
- Métricas: número de clases por paquete, número de líneas de código exceptuando comentarios, número de bloques Javadoc, líneas de documentación, complejidad ciclomática, bloques de líneas duplicadas, responsabilidad, independencia, nivel de abstracción, inestabilidad, dependencias cíclicas.

3. Diseño técnico

En este apartado, se especifica el diseño e implementación de cada uno de los módulos implementados en el sistema. En los capítulos anteriores, se ha ofrecido una breve introducción de cada uno de ellos, así como de los componentes externos que se utilizan; en este apartado se detalla la funcionalidad de cada uno de ellos especificando el procesamiento que se realiza en el sistema.

La siguiente ilustración muestra un esquema del flujo de la aplicación:

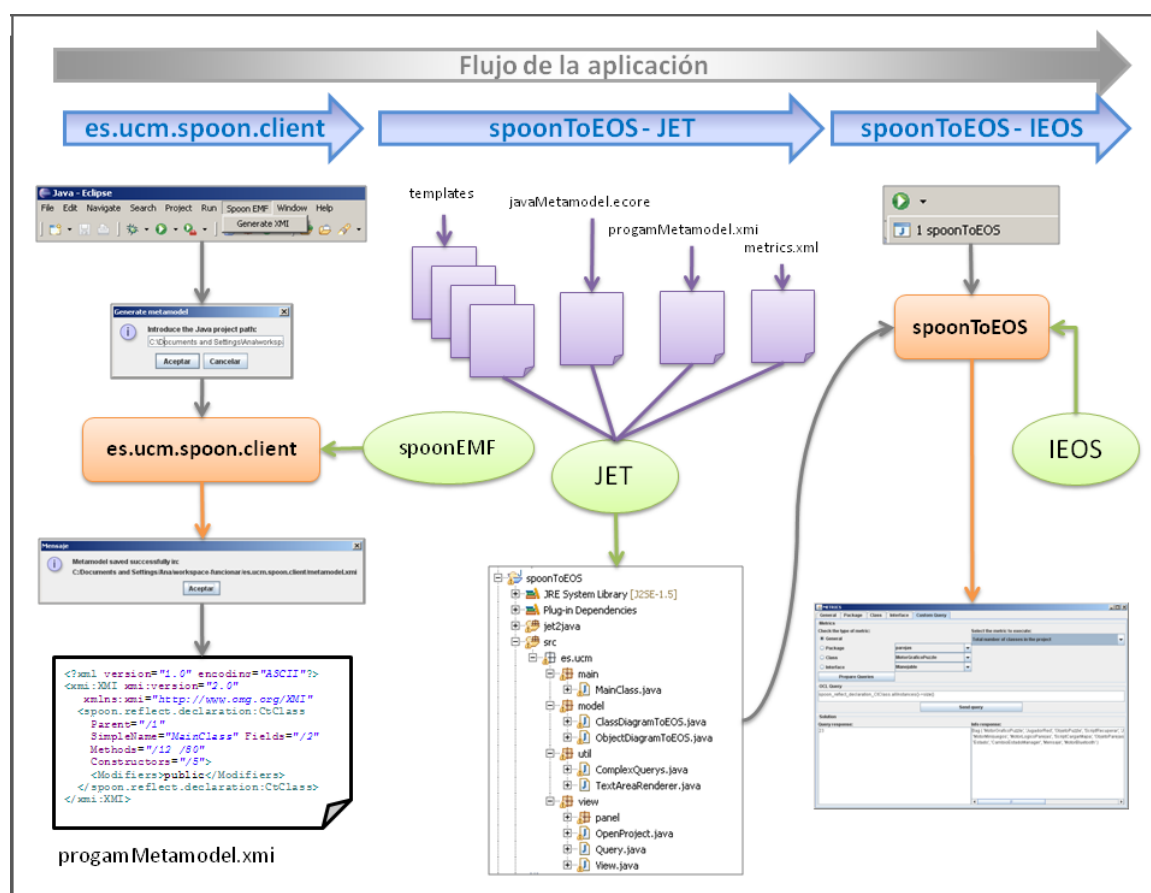


Ilustración 17: Diseño técnico: Flujo de la aplicación.

3.1 Módulo es.ucm.spoon.client

3.1.1 Descripción

El módulo **es.ucm.spoon.client** se encarga de generar el metamodelo de un determinado programa Java. Este metamodelo es una instancia del metamodelo de Java y se almacena en un fichero de salida cuyo formato es *.xmi.

3.1.2 Tipo

Este módulo es un plugin desarrollado en Eclipse que internamente hace referencia a todas las librerías incluidas en la herramienta spoonEMF para implementar el mismo comportamiento que esta herramienta.

El plugin es.ucm.spoon.client añade un nuevo menú a la barra de menús de eclipse, el nuevo menú se denomina **Spoon EMF** y a su vez, contiene el submenú **Generate XMI**. La siguiente ilustración muestra el plugin instalado en el eclipse:

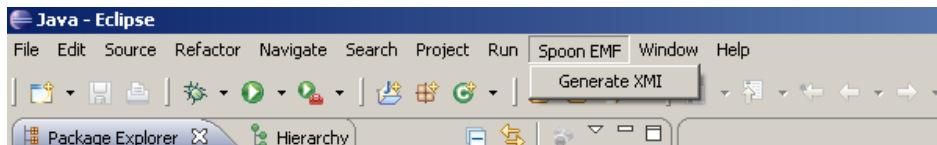


Ilustración 18: Diseño técnico: Plugin es.ucm.spoon.client

3.1.3 Estructura

La siguiente ilustración muestra la estructura del código fuente del plugin es.ucm.spoon.client:

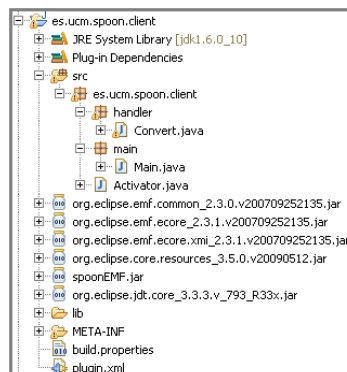


Ilustración 19: Diseño técnico: Estructura es.ucm.spoon.client

El plugin **es.ucm.spoon.client** está estructurado de la siguiente manera: el paquete **es.ucm.spoon.client.handler** almacena la clase **Convert.java**, que es la clase que se ejecuta cuando el usuario selecciona el submenú “Generate XMI”; el paquete **es.ucm.spoon.client.main** almacena la clase **Main.java** que invoca a **spoonEMF** para generar el modelo de un programa Java determinado y finalmente la clase **Activator.java** almacenada directamente en el paquete **es.ucm.spoon.client** que controla el ciclo de vida del plugin.

El fichero plugin.xml es donde se define el tipo de plugin y las extensiones que utiliza, para este plugin se ha utilizado la extensión que ofrece Eclipse para añadir nuevos menús en `org.eclipse.ui.menus`. La siguiente ilustración muestra el fichero plugin.xml de `es.ucm.spoon.client`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
  <extension
    point="org.eclipse.ui.menus">
    <menuContribution
      locationURI="menu:org.eclipse.ui.main.menu">
      <menu label="Spoon EMF">
        <command
          commandId="es.ucm.spoon.client.command"
          label="Generate XMI"
          style="push">
        </command>
      </menu>
    </menuContribution>
  </extension>
  <extension
    point="org.eclipse.ui.handlers">
    <handler
      class="es.ucm.spoon.client.handler.Convert"
      commandId="es.ucm.spoon.client.command">
    </handler>
  </extension>
  <extension
    point="org.eclipse.ui.commands">
    <command
      id="es.ucm.spoon.client.command"
      name="Generate XMI">
    </command>
  </extension>
</plugin>
```

Ilustración 20: Diseño técnico: Definición del plugin `es.ucm.spoon.client` en plugin.xml

3.1.4 Procesamiento

La funcionalidad del plugin `es.ucm.spoon.client` está definida de la siguiente manera:

- El usuario selecciona la opción **Generate XMI**.
- Se pregunta al usuario por la ruta donde está almacenado el programa Java y el usuario introduce la ruta completa del programa cuyo metamodelo desea exportar.

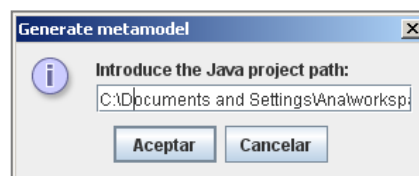


Ilustración 21: Diseño técnico: `es.ucm.spoon.client` Introduce the Java Project path.

- El plugin genera el metamodelo del proyecto introducido invocando al método `main` de la clase `main.Java2XMIHelper` de la librería `spoonEMF.jar` y le pasa como parámetros de entrada el fichero de salida donde se almacena el metamodelo y la ruta del programa Java. Se ha implementado que el fichero de salida se denomine `metamodel.xml` y se almacena en la carpeta `es.ucm.spoon.client` ubicada en el directorio de trabajo o workspace del Eclipse.
- Finalmente, el plugin muestra un mensaje al usuario indicando que el modelo se ha generado correctamente y muestra la ruta donde se ha almacenado el fichero de salida `metamodel.xml`.

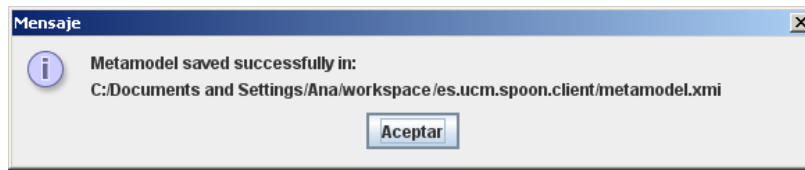


Ilustración 22: Diseño técnico: es.ucm.spoon.client Metamodel saved successfully.

3.2 Módulo SpoonToEOS

3.2.1 Descripción

El módulo **SpoonToEOS** se puede considerar como el núcleo del sistema. Es el componente donde se genera el código de la aplicación y por tanto, el componente que se ejecuta para lanzar la aplicación cliente y ejecutar las métricas definidas sobre un programa Java concreto.

3.2.2 Tipo

Este componente es un proyecto JET de Eclipse y por tanto, este proyecto está preparado para poder exportarse como un plugin; se especifica más información en el capítulo [Conclusiones](#).

3.2.3 Estructura

La siguiente ilustración muestra la estructura del componente **spoonToEOS**:

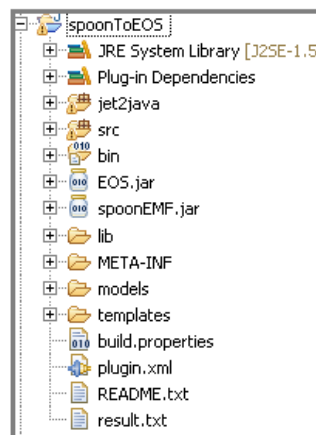


Ilustración 23: Diseño técnico: Estructura spoonToEOS

El proyecto JET spoonToEOS está estructurado de la siguiente manera:

- La carpeta **models** está destinada a almacenar los ficheros que contienen los parámetros de entrada para generar el código fuente de la aplicación, concretamente almacena los siguientes ficheros: **javaMetamodel.ecore**, que corresponde al metamodelo de Java; **metamodel.xmi** que se corresponde con la instancia del metamodelo de Java generada por el plugin es.ucm.spoon.client especificado en el punto anterior [Módulo es.ucm.spoon.client](#) y el fichero **metrics.xml** que almacena las métricas definidas en la aplicación con su correspondiente expresión OCL.

- La carpeta **templates** almacena las plantillas *.jet que se utilizan para generar el código de la aplicación: la plantilla **main.jet** es la plantilla inicial que va invocando al resto de plantillas para generar cada uno de los paquetes y clases que forman el código fuente de la aplicación Java resultante.
- La carpeta **lib** almacena las librerías necesarias para la ejecución de la aplicación: **spoonEMF.jar** para reconocer el metamodelo generado por el plugin es.ucm.spoon.client y la librería **EOS.jar** para la evaluación de métricas.
- La carpeta **jet2java** es una carpeta autogenerada por JET y almacena el código autogenerado por JET previo a la generación del código fuente de la aplicación, es decir, son las clases resultantes del procesamiento de cada una de las templates y se encargan de “escribir” las clases *.java resultantes.
- La carpeta **src** es una carpeta autogenerada por JET, está especificada en la plantilla inicial main.jet y se corresponde con la carpeta fuente de la aplicación final, es decir, donde se almacena el código fuente resultante de spoonToEOS.

3.2.4 Procesamiento

El procesamiento del módulo spoonToEOS se puede dividir en dos fases: el proceso que ejecuta JET para generar el código fuente a partir de las plantillas definidas en la carpeta templates y por otro lado, el proceso de evaluar las métricas utilizando la aplicación que se lanza en la ejecución del código fuente resultante.

Todo proyecto JET de Eclipse define sus propiedades en el fichero de configuración plugin.xml, concretamente este tipo de proyectos, utilizan como extensión la librería org.eclipse.jet.transform de eclipse, a continuación se muestran las propiedades definidas para este componente:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
  <extension
    point="org.eclipse.jet.transform">
    <transform
      enableEmbeddedExpressions="true"
      modelLoader="org.eclipse.jet.emf"
      startTemplate="templates/main.jet"
      templateLoaderClass="org.eclipse.jet.compiled._jet_transformation">
      <description></description>
      <tagLibraries>
        <importLibrary id="org.eclipse.jet.controlTags" usePrefix="c" autoImport="true"/>
        <importLibrary id="org.eclipse.jet.javaTags" usePrefix="java" autoImport="true"/>
        <importLibrary id="org.eclipse.jet.formatTags" usePrefix="f" autoImport="true"/>
        <importLibrary id="org.eclipse.jet.workspaceTags" usePrefix="ws" autoImport="false"/>
      </tagLibraries>
    </transform>
  </extension>
</plugin>
```

Ilustración 24: Diseño técnico: Propiedades JET de spoonToEOS.

Como puede apreciarse en la ilustración anterior, la plantilla inicial para comenzar el proceso de generación de código es main.jet; esta plantilla es la encargada de crear la estructura del código fuente resultante y de establecer las plantillas que deben procesarse para crear cada una de las clases. El [Apéndice C](#) muestra el contenido de la plantilla main.jet.

Generación de código

El código fuente se genera en la carpeta fuente src:

- El paquete es.ucm.main contiene la clase MainClass.java, esta clase contiene el método main de la aplicación y se genera a partir de la plantilla **main_process.java.jet**.
- El paquete es.ucm.model contiene las clases encargadas de insertar el metamodelo de Java como diagrama de clases y la instancia del metamodelo de Java como diagrama de objetos en el componente IEOS. Estas clases son ClassDiagramToEOS.java y ObjectDiagramToEOS.java respectivamente.

- La clase ClassDiagramToEOS.java se genera a partir de la plantilla **ecore_procces.java.jet**. Esta plantilla carga como parámetro de entrada el fichero javaMetamodel.ecore almacenado en la carpeta models, puesto que es el fichero que almacena el metamodelo de Java:

```
<c:load url="models/javaMetamodel.ecore" var="ePackage"
loader="org.eclipse.jet.emf"/>
```

La plantilla ecore_procces.java.jet almacena el nodo raíz en la variable ePackage y va procesando todos los nodos hijos para ir formando el código que se necesita para insertar el diagrama de clases en EOS tal y como se especifica en su API [<http://maude.sip.ucm.es/eos/v0.3/doc/index.html>].

- La clase ObjectDiagramToEOS.java se genera a partir de la plantilla **spoon_report.java.jet**. Esta plantilla carga como parámetro de entrada el fichero metamodel.xmi, que es el fichero generado por el plugin es.ucm.spoon.client al exportar el metamodelo de un determinado programa Java:

```
<c:load url="models/metamodel.xmi" var="spoon"
loader="org.eclipse.jet.emfxml"/>
```

La plantilla spoon_report.java.jet almacena el nodo raíz en la variable spoon y va procesando todos los nodos hijos para ir formando el código que se necesita para insertar el diagrama de objetos en EOS tal y como se especifica en su API [<http://maude.sip.ucm.es/eos/v0.3/doc/index.html>].

- El paquete es.ucm.view almacena las clases encargadas de implementar la vista de la aplicación. La clase OpenProject.java que crea el panel principal y se genera a partir de la plantilla **open_project.java.jet**. La clase View.java que crea todos los paneles que están incluidos en el panel principal e inicializa todas las queries especificadas en el fichero metrics.xml. La clase View.java se genera a partir de la plantilla **gui.java.jet** y esta plantilla carga como parámetro de entrada el fichero metrics.xml, al ser esta clase la encargada de procesar el fichero metrics.xml y mapear cada una de las métricas definidas en objetos de tipo Query.

```
<c:load url="models/metrics.xml" var="metrics" loader="org.eclipse.jet.emfxml"/>
```

La clase Query.java es una entidad que se utiliza para almacenar una determinada métrica definida en el fichero metrics.xml. La aplicación almacena todas las métricas en un mapa de objetos de tipo Query para facilitar el acceso a las mismas cuando el usuario final seleccione cualquier métrica que desea ejecutar en un momento dado. La clase Query.java se genera a partir de la plantilla **query.java.jet**.

El paquete es.ucm.view contiene a su vez, el paquete es.ucm.view.panel que almacena las clases que implementan cada uno de los paneles de la interfaz de la aplicación:

- MetricsPanel.java especifica el panel que contiene la tabla de resultados y se genera a partir de la plantilla **metrics_panel.java.jet**.
 - SpecificPanel.java especifica el panel que permite al usuario ejecutar las métricas de paquetes, clases o interfaces y se genera a partir de la plantilla **specific_panel.java.jet**.
 - GeneralPanel.java especifica el panel que permite al usuario visualizar las métricas generales del proyecto, esta clase se genera a partir de la plantilla **general_panel.java**.
 - Y finalmente CustomPanel.java que especifica el panel personalizable que permite al usuario ejecutar indistintamente cualquiera de las métricas definidas en la aplicación o introducir nuevas métricas con expresiones OCL y ejecutarlas al instante, esta clase se genera a partir de la plantilla **custom_panel.java.jet**.
- El paquete es.ucm.view almacena las clases que implementan utilidades específicas de la aplicación:
 - ComplexQueries.java que almacena las métricas complejas que no se especifican a partir de más de una expresión OCL, la plantilla **complex_queries.java.jet** es la encargada de generar esta clase.
 - La clase TextAreaRendered.java necesaria para escribir dinámicamente en las celdas de las tablas, la plantilla **textarea_rendered.java.jet** especifica cómo generar esta clase.

Ejecución de la aplicación

Una vez que se ha generado el código fuente de la aplicación, se puede ejecutar la aplicación spoonToEOS como una aplicación Java.

Inicialización del evaluador

Las operaciones iniciales que realiza la aplicación son las siguientes:

1. Crear una instancia del evaluador IEOS.
2. Insertar el diagrama de clases en el evaluador, la clase ClassDiagramToEOS es la encargada de realizar esta tarea.

3. Insertar el diagrama de objetos en el evaluador, la clase `ObjectDiagramToEOS` es la encargada de realizar esta tarea.
4. Lanzar la interfaz de usuario invocando a `OpenProject` con la instancia del evaluador creada en los tres pasos anteriores.

Procesamiento de métricas

En la inicialización de la interfaz de usuario de la aplicación, se incluye también la inicialización de cada uno de los conjuntos de métricas: el conjunto de métricas generales, el conjunto de métricas asociadas a paquetes, el conjunto de métricas asociadas a clases y el conjunto de métricas asociadas a interfaces. Cada uno de estos conjuntos es de tipo `Map<String, Query>` donde la clave de cada entrada es el título de la métrica y `Query` es el objeto que contiene toda la información necesaria para ejecutar una determinada métrica.

El esquema XML del fichero `metrics.xml` es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- ROOT -->
  <xsd:element name="metrics">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="group" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- One element by metrics group, initially: general, package, classes and
  interfaces. -->
  <xsd:element name="group">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="metric" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <!-- One element by metric on each group -->
  <xsd:element name="metric">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="operation" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="info" type="xsd:string" minOccurs="0"
          maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="title" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <!-- Specify the OCL operation or the method needed to be executed -->
  <xsd:element name="operation" type="xsd:string">
    <xsd:complexType>
      <xsd:attribute name="type" type="metricType" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <!-- Specify if the type attribute of operation element is a OCL expression
  or the name of the method to execute in ComplexQueries class -->
  <xsd:simpleType name="metricType">
    <xsd:restriction>
      <xsd:enumeration value="ocl" />
      <xsd:enumeration value="method" />
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

El elemento `<metrics>` es el nodo raíz de la estructura XML y contiene elementos `<group>`, este elemento agrupa las métricas en función del elemento Java sobre el que se han definido, inicialmente se han definido métricas generales que evalúan todo el proyecto, métricas que se evalúan dentro de un determinado paquete, métricas que se evalúan dentro de una determinada clase y métricas que se evalúan dentro de una determinada interfaz, por tanto se ha definido un elemento `<group>` por cada uno de estos conjuntos de métricas indicando en su atributo `id` a qué grupo pertenecen: `general`, `package`, `class` o `interface`.

Cada métrica se especifica en el elemento `<metric>`, cuyo atributo `title` indica el título de la métrica en cuestión, este título se utiliza para mostrar al usuario cada una de las posibles métricas que se permiten ejecutar. Además el elemento `<metric>`, contiene el elemento `<operation>` y el elemento `<info>`.

El elemento `<operation>` indica la operación que debe ejecutarse para evaluar dicha métrica, este elemento contiene un atributo `type` que indica el tipo de operación: operaciones de tipo `ocl` que almacenan la expresión OCL correspondiente a la métrica especificada en el elemento `<metric>`, y operaciones de tipo `method`, que almacenan el nombre del método que se debe ejecutar para evaluar la métrica en cuestión. Este último tipo de métricas se utiliza para aquellas métricas que no se han podido obtener a partir de una sola expresión OCL, sino que necesitan una lógica adicional, concretamente se utiliza este tipo de operaciones para aquellas métricas que implican procesamiento recursivo. En general, también facilitan el uso de expresiones OCL predefinidas.

El elemento `<info>` es opcional, y se utiliza para especificar una segunda expresión OCL que ofrece más información al resultado de la expresión OCL del elemento `<operation>`. Por ejemplo, la métrica que calcula el número de clases, almacena como operación principal la expresión OCL que obtiene el número total de clases y una segunda expresión OCL que obtiene el nombre de cada una de ellas como información adicional.

En total se han definido cuarenta y nueve métricas. Las métricas especificadas en el fichero `metrics.xml` son las siguientes:

- **Métricas generales**, se han definido veintitrés métricas:

Número total de paquetes	
Operación	<code>spoon_reflect_declaration_CtPackage.allInstances()->size()</code>
Información	<code>spoon_reflect_declaration_CtPackage.allInstances().SimpleName</code>
Número total de clases	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName</code>
Número total de interfaces	
Operación	<code>spoon_reflect_declaration_CtInterface.allInstances()->size()</code>
Información	<code>spoon_reflect_declaration_CtInterface.allInstances().SimpleName</code>
Número total de atributos	
Operación	<code>spoon_reflect_declaration_CtField.allInstances()->size()</code>

Información	<code>spoon_reflect_declaration_CtField.allInstances().SimpleName</code>
Número total de métodos	
Operación	<code>spoon_reflect_declaration_CtMethod.allInstances()->size()</code>
Información	<code>spoon_reflect_declaration_CtMethod.allInstances().SimpleName</code>
Número total de clases que heredan de otra clase padre	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(class class.Superclass->size()>=1)->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(class class.Superclass->size()>=1).SimpleName</code>
Número total de clases que son clases padre de otras clases	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().Superclass->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances().Superclass.SimpleName</code>
Número total de expresiones if	
Operación	<code>spoon_reflect_code_CtIf.allInstances()->size()</code>
Información	N/A
Número total de bucles for	
Operación	<code>spoon_reflect_code_CtFor.allInstances()->size()</code>
Información	N/A
Número total de bucles while	
Operación	<code>spoon_reflect_code_CtWhile.allInstances()->size()</code>
Información	N/A
Media de clases por paquete	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->size().div(spoon_reflect_declaration_CtPackage.allInstances()->size())</code>
Información	N/A
Media de atributos por clase	
Operación	<code>spoon_reflect_declaration_CtField.allInstances()->size().div(spoon_reflect_declaration_CtClass.allInstances()->size())</code>
Información	N/A
Media de métodos por clase	
Operación	<code>spoon_reflect_declaration_CtMethod.allInstances()->size().div(spoon_reflect_declaration_CtClass.allInstances()->size())</code>
Información	N/A
Número total de clases que son utilizadas como tipos de atributos	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_declaration_CtField.allInstances().Type.SimpleName->includes(x))->size()</code>

Información	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_declaration_CtField.allInstances().Type.SimpleName->includes(x))</code>
Número total de clases que son utilizadas como tipos de variables locales	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_code_CtLocalVariable.allInstances().Type.SimpleName->includes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_code_CtLocalVariable.allInstances().Type.SimpleName->includes(x))</code>
Número total de clases que no son referenciadas en el resto del proyecto	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))</code>
Número total de interfaces que no son referenciadas en el resto del proyecto	
Operación	<code>spoon_reflect_declaration_CtInterface.allInstances().SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtInterface.allInstances().SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))</code>
Número total de interfaces que no son implementadas por ninguna clase	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().Superinterfaces.SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances().Superinterfaces.SimpleName->select(x spoon_reflect_reference_CtTypeReference.allInstances().SimpleName->excludes(x))</code>
Número total de paquetes que no son referenciados en el resto del proyecto	
Operación	<code>spoon_reflect_declaration_CtPackage.allInstances().SimpleName->select(x spoon_reflect_reference_CtPackageReference.allInstances().SimpleName->excludes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtPackage.allInstances().SimpleName->select(x spoon_reflect_reference_CtPackageReference.allInstances().SimpleName->excludes(x))</code>
Profundidad del árbol de herencia	
Operación	<code>getDepthInheritanceTree</code>
Información	N/A
Número total de clases que son utilizadas como tipos de parámetros	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances().SimpleName->select(x spoon_reflect_declaration_CtParameter.allInstances().Type.SimpleName->includes(x))->size()</code>

Información	<pre> spoon_reflect_declaration_CtClass.allInstances().SimpleName-> select(x spoon_reflect_declaration_CtParameter.allInstances().Type.SimpleName-> includes(x)) </pre>
Número total de atributos cuyo tipo se corresponde con una clase o interfaz	
Operación	<pre> spoon_reflect_declaration_CtField.allInstances().Type.SimpleName-> select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName-> includes(x))->size() </pre>
Información	<pre> spoon_reflect_declaration_CtField.allInstances()-> select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName-> includes(x.Type.SimpleName)).SimpleName </pre>
Número total de parámetros cuyo tipo se corresponde con una clase o interfaz	
Operación	<pre> spoon_reflect_declaration_CtParameter.allInstances().Type.SimpleName-> select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName-> includes(x))->size() </pre>
Información	<pre> spoon_reflect_declaration_CtParameter.allInstances()-> select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName-> includes(x.Type.SimpleName)).SimpleName </pre>

Tabla 1: Diseño técnico: Métricas generales.

- **Métricas por paquete**, se han definido cinco métricas:

En este conjunto de métricas cabe destacar que la cadena '{value}' que aparece en las expresiones OCL se sustituye por el nombre del paquete seleccionado por el usuario a través de la interfaz de usuario.

Número de clases dentro del paquete seleccionado	
Operación	<pre> spoon_reflect_declaration_CtPackage.allInstances()-> select(p p.SimpleName='{value}').Types-> select(i i.oclIsTypeOf(spoon_reflect_declaration_CtClass)=true)->size() </pre>
Información	<pre> spoon_reflect_declaration_CtPackage.allInstances()-> select(p p.SimpleName='{value}').Types-> select(i i.oclIsTypeOf(spoon_reflect_declaration_CtClass)=true).SimpleName </pre>
Número total de clases dentro del paquete seleccionado incluyendo los subpaquetes	
Operación	<pre> getNumberOfClassInAPackage </pre>
Información	<pre> getClassInAPackage </pre>
Número de interfaces dentro del paquete seleccionado	
Operación	<pre> spoon_reflect_declaration_CtPackage.allInstances()-> select(p p.SimpleName='{value}').Types-> select(i i.oclIsTypeOf(spoon_reflect_declaration_CtInterface)=true)->size() </pre>
Información	<pre> spoon_reflect_declaration_CtPackage.allInstances()-> select(p p.SimpleName='{value}').Types-> select(i i.oclIsTypeOf(spoon_reflect_declaration_CtInterface)=true).SimpleName </pre>
Número total de interfaces dentro del paquete seleccionado incluyendo los subpaquetes	
Operación	<pre> getNumberOfInterfacesInAPackage </pre>
Información	<pre> getInterfacesInAPackage </pre>

Número de veces que un elemento del paquete es referenciado en el resto del proyecto	
Operación	<code>spoon_reflect_reference_CtPackageReference.allInstances()->select(p p.SimpleName.size()>{size}).SimpleName->select(s s.substring(s.size()-{size}+1,s.size())='{value}')->size()</code>
Información	N/A

Tabla 2: Diseño técnico: Métricas por paquete.

- **Métricas por clase**, se han definido quince métricas:

En este conjunto de métricas cabe destacar que la cadena '`{value}`' que aparece en las expresiones OCL se sustituye por el nombre de la clase seleccionada por el usuario a través de la interfaz de usuario.

Número total de atributos dentro de la clase seleccionada	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(p p.SimpleName='{value}').Fields->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(p p.SimpleName='{value}').Fields.SimpleName</code>
Número total de métodos definidos dentro de la clase seleccionada	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(p p.SimpleName='{value}').Methods->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(p p.SimpleName='{value}').Methods.SimpleName</code>
Número total de clases hijas de la clase seleccionada	
Operación	<code>spoon_reflect_reference_CtTypeReference.allInstances()->select(t t.Superclass_2->size()>0)->select(r r.SimpleName='{value}').Superclass_2->size()</code>
Información	<code>spoon_reflect_reference_CtTypeReference.allInstances()->select(t t.Superclass_2->size()>0)->select(r r.SimpleName='{value}').Superclass_2.SimpleName</code>
Número total de clases padre directas de la clase seleccionada	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Superclass->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Superclass.SimpleName</code>
Número total de interfaces que implementa la clase seleccionada	
Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Superinterfaces->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Superinterfaces.SimpleName</code>
Número total de atributos heredados	
Operación	<code>getNumberInheritedAttributes</code>
Información	<code>getInheritedAttributes</code>
Número total de métodos heredados	

Operación	getNumberInheritedOperations
Información	getInheritedOperations
Número total de veces que la clase seleccionada es utilizada como un tipo de atributo	
Operación	spoon_reflect_declaration_CtField.allInstances().Type->select(c c.SimpleName='{value}')->size()
Información	N/A
Número de veces que la clase seleccionada es utilizada como tipo de parámetros	
Operación	spoon_reflect_declaration_CtParameter.allInstances().Type->select(c c.SimpleName='{value}')->size()
Información	N/A
Número de atributos en la clase seleccionada que tienen como tipo una interfaz del proyecto	
Operación	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Fields.Type.SimpleName->select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x))->size()
Información	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Fields->select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName
Número de atributos en la clase seleccionada que tienen como tipo otra clase del proyecto	
Operación	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Fields.Type.SimpleName->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x))->size()
Información	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Fields->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName
Número de atributos locales que no son referenciados en la clase	
Operación	spoon_reflect_declaration_CtField.allInstances()->select(x x.DeclaringType.SimpleName='{value}').SimpleName->select(y spoon_reflect_reference_CtFieldReference.allInstances().SimpleName->excludes(y))->size()
Información	spoon_reflect_declaration_CtField.allInstances()->select(x x.DeclaringType.SimpleName='{value}').SimpleName->select(y spoon_reflect_reference_CtFieldReference.allInstances().SimpleName->excludes(y))
Número de parámetros de entrada en la clase seleccionada que tienen como tipo una interfaz del proyecto	
Operación	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters.Type.SimpleName->select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x))->size()
Información	spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters->select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName
Número de parámetros de entrada en la clase seleccionada que tienen como tipo otra clase del proyecto	

Operación	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters.Type.SimpleName->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtClass.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName</code>
Falta de cohesión en la clase seleccionada	
Operación	<code>getLackOfCohesion</code>
Información	N/A

Tabla 3: Diseño técnico: Métricas por clase.

- **Métricas por interfaz**, se han definido seis métricas:

En este conjunto de métricas cabe destacar que la cadena '{value}' que aparece en las expresiones OCL se sustituye por el nombre de la interfaz seleccionada por el usuario a través de la interfaz de usuario.

Número de métodos definidos en la interfaz seleccionada	
Operación	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods->size()</code>
Información	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods.SimpleName</code>
Número de clases que implementan directamente la interfaz seleccionada	
Operación	<code>spoon_reflect_reference_CtTypeReference.allInstances()->select(t t.Superinterfaces_2->size()>0)->select(r r.SimpleName='{value}').Superinterfaces_2->size()</code>
Información	<code>spoon_reflect_reference_CtTypeReference.allInstances()->select(t t.Superinterfaces_2->size()>0)->select(r r.SimpleName='{value}').Superinterfaces_2.SimpleName</code>
Número de veces que la interfaz seleccionada es utilizada como tipo de atributo	
Operación	<code>spoon_reflect_declaration_CtField.allInstances().Type->select(c c.SimpleName='{value}')->size()</code>
Información	N/A
Número de veces que la interfaz seleccionada es utilizada como tipo de parámetro	
Operación	<code>spoon_reflect_declaration_CtParameter.allInstances().Type->select(c c.SimpleName='{value}')->size()</code>
Información	N/A
Número de parámetros en la interfaz seleccionada que tienen como tipo otra interfaz del proyecto	
Operación	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters.Type.SimpleName->select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters-></code>

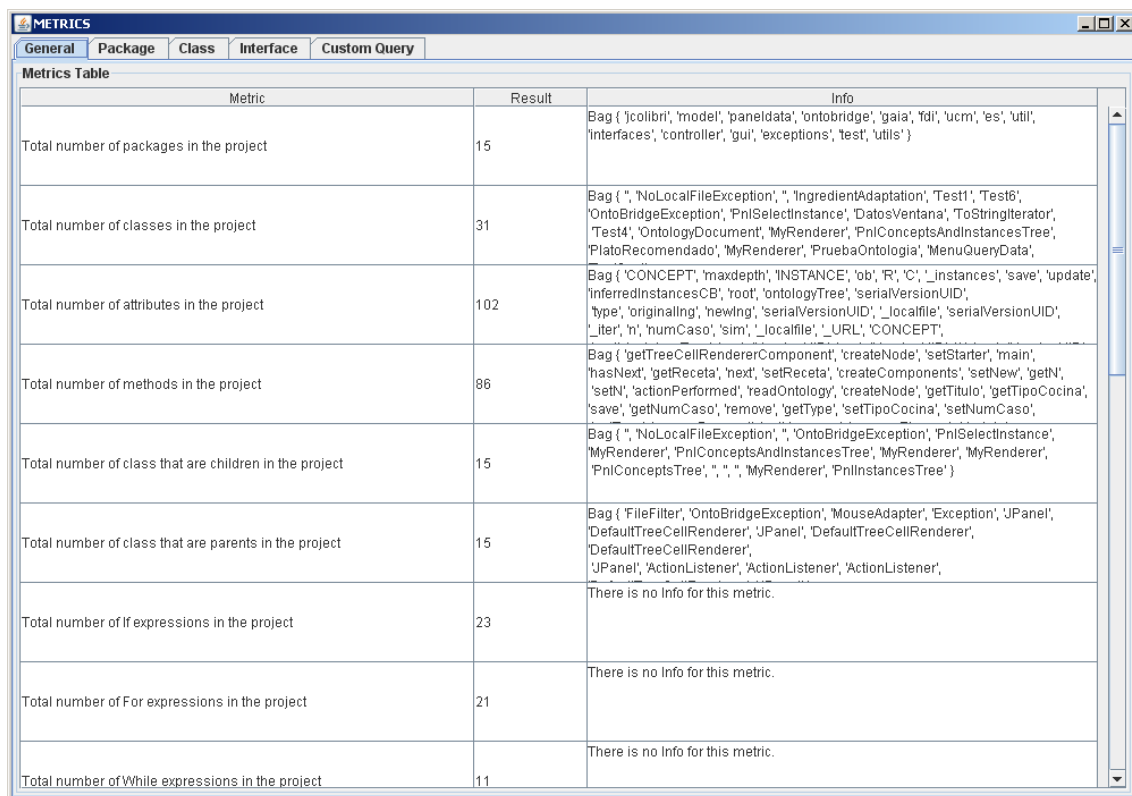
	<code>select(x spoon_reflect_declaration_CtInterface.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName</code>
Número de parámetros en la interfaz seleccionada que tienen como tipo una clase del proyecto	
Operación	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters.Type.SimpleName->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x))->size()</code>
Información	<code>spoon_reflect_declaration_CtInterface.allInstances()->select(c c.SimpleName='{value}').Methods.Parameters->select(x spoon_reflect_declaration_CtClass.allInstances().SimpleName->includes(x.Type.SimpleName)).SimpleName</code>

Tabla 4: Diseño técnico: Métricas por interfaz.

Utilidades de la aplicación

La interfaz de usuario está formada por cuatro pestañas:

- **General:** esta pestaña muestra al usuario una tabla con todas las métricas generales, especificando el título, el resultado y la información adicional de cada una de ellas.



Metric	Result	Info
Total number of packages in the project	15	Bag { 'colibri', 'model', 'paneldata', 'ontobridge', 'gaia', 'fdi', 'ucm', 'es', 'util', 'interfaces', 'controller', 'gui', 'exceptions', 'test', 'utils' }
Total number of classes in the project	31	Bag { 'NoLocalFileException', 'IngredientAdaptation', 'Test1', 'Test6', 'OntoBridgeException', 'PnlSelectInstance', 'DatosVentana', 'ToStringIterator', 'Test4', 'OntologyDocument', 'MyRenderer', 'PnlConceptsAndInstancesTree', 'PlatoRecomendado', 'MyRenderer', 'PruebaOntologia', 'MenuQueryData', 'PnlConceptsTree', 'PnlInstancesTree' }
Total number of attributes in the project	102	Bag { 'CONCEPT', 'maxdepth', 'INSTANCE', 'ob', 'R', 'C', '_instances', 'save', 'update', 'InferredInstancesCB', 'root', 'ontologyTree', 'serialVersionUID', 'type', 'originalling', 'newling', 'serialVersionUID', '_localfile', 'serialVersionUID', '_iter', 'n', 'humCaso', 'sim', '_localfile', '_URL', 'CONCEPT' }
Total number of methods in the project	86	Bag { 'getTreeCellRendererComponent', 'createNode', 'setStarter', 'main', 'hasNext', 'getReceta', 'next', 'setReceta', 'createComponents', 'setNew', 'getN', 'setN', 'actionPerformed', 'readOntology', 'createNode', 'getTitulo', 'getTipoCocina', 'save', 'getNumCaso', 'remove', 'getType', 'setTipoCocina', 'setNumCaso' }
Total number of class that are children in the project	15	Bag { 'NoLocalFileException', 'OntoBridgeException', 'PnlSelectInstance', 'MyRenderer', 'PnlConceptsAndInstancesTree', 'MyRenderer', 'MyRenderer', 'PnlConceptsTree', 'PnlInstancesTree' }
Total number of class that are parents in the project	15	Bag { 'FileFilter', 'OntoBridgeException', 'MouseAdapter', 'Exception', 'JPanel', 'DefaultTreeCellRenderer', 'JPanel', 'DefaultTreeCellRenderer', 'JPanel', 'ActionListener', 'ActionListener', 'ActionListener' }
Total number of If expressions in the project	23	There is no Info for this metric.
Total number of For expressions in the project	21	There is no Info for this metric.
Total number of While expressions in the project	11	There is no Info for this metric.

Ilustración 25: Diseño técnico: Pestaña General.

- **Package:** esta pestaña permite al usuario seleccionar un paquete determinado de entre todos los paquetes existentes en la aplicación Java que se está evaluando.

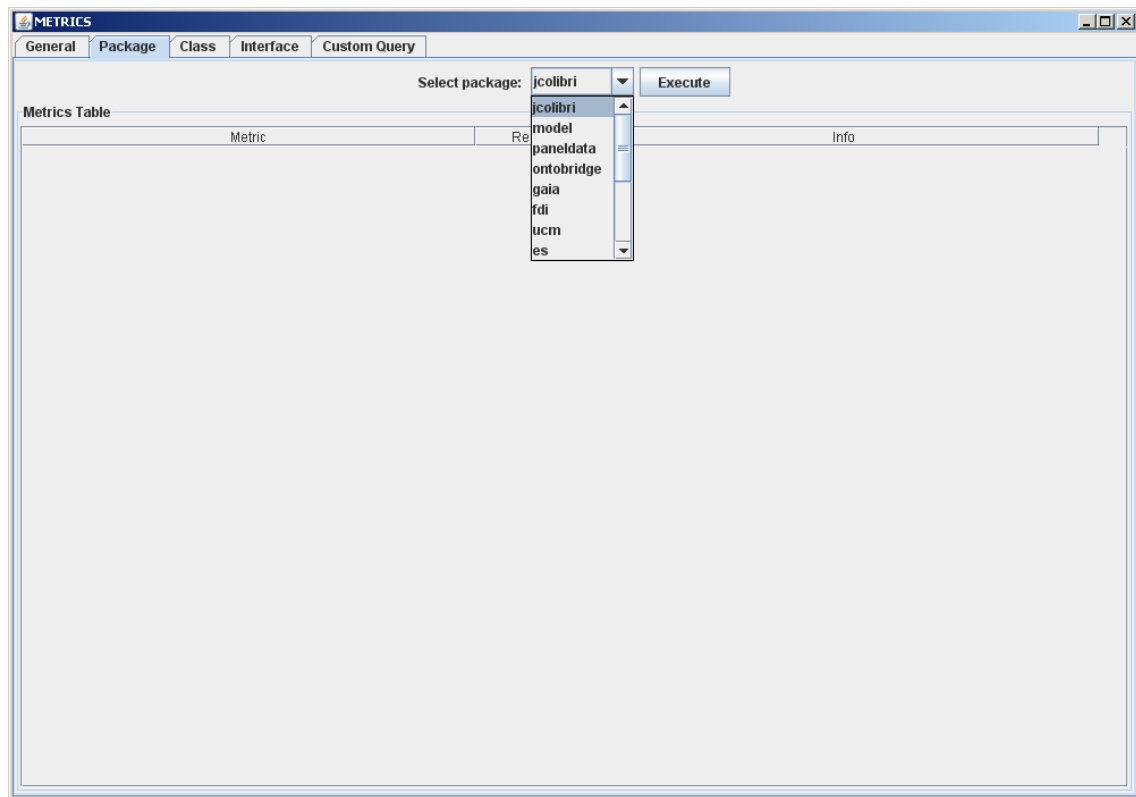


Ilustración 26: Diseño técnico: Pestaña Package, seleccionar un paquete.

Una vez que el usuario selecciona un determinado paquete y pulsa el botón “*Execute*” se muestra una tabla con toda la información resultante al evaluar las métricas correspondientes al grupo “package” sobre el paquete que ha seleccionado el usuario.

Metric	Result	Info
Number of Classes in a package	0	Bag { }
Number of Classes in a package and its subpackages	5	„DatosVentana“, „MenuQueryData“, „PlatoRecomendado“, „SingleQueryData“
Number of Interfaces in a package	0	Bag { }
Number of Interfaces in a package and its subpackages	6	„InterfazElementosModelo“, „InterfazModelo“, „InterfazQuery“, „InterfazRecomendacion“, „InterfazVista“
Number of times an element of the package is referenced in the project	0	There is no Info for this metric.

Ilustración 27: Diseño técnico: Pestaña Package, evaluar métricas.

- **Class:** esta pestaña permite al usuario seleccionar una clase determinada de entre todas las clases existentes en la aplicación Java que se está evaluando.

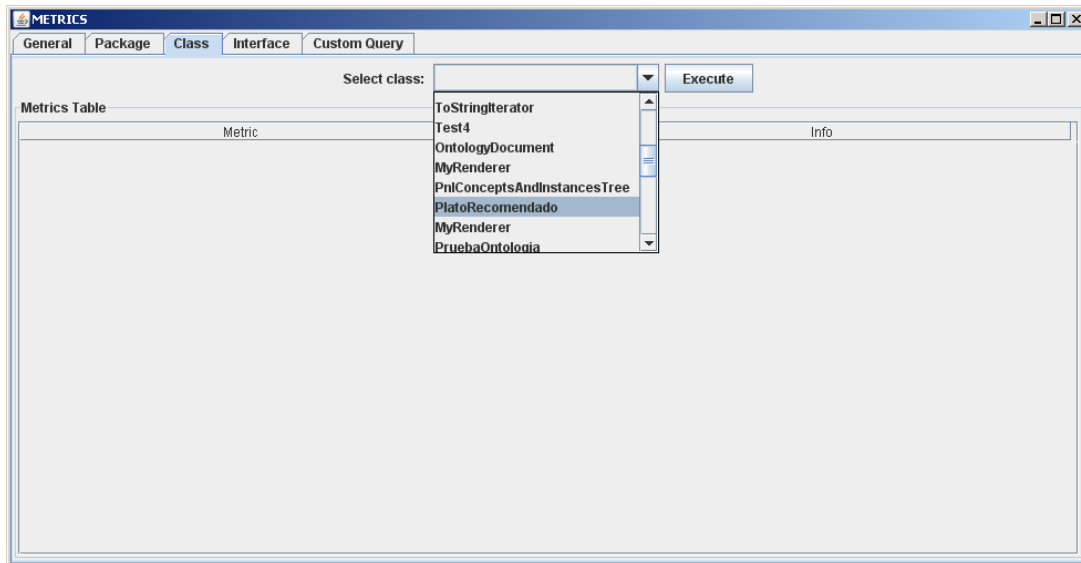


Ilustración 28: Diseño técnico: Pestaña Class, seleccionar una clase.

Una vez que el usuario selecciona una determinada clase y pulsa el botón “Execute” se muestra una tabla con toda la información resultante al evaluar las métricas correspondientes al grupo “class” sobre la clase que ha seleccionado el usuario.

Metric	Result	Info
Total number of attributes in a class	10	Bag { 'ANY', 'STARTER', 'MAIN', 'DESSERT', 'ingredientesSi', 'ingredientesNo', 'dieta', 'tipoCocina', 'tipoPlato', 'platoMenu' }
Total number of methods in a class	12	Bag { 'getIngredientesSi', 'setIngredientesSi', 'getIngredientesNo', 'setIngredientesNo', 'getDieta', 'setDieta', 'getPlatoMenu', 'setPlatoMenu', 'getTipoCocina', 'setTipoCocina', 'getTipoPlato', 'setTipoPlato' }
Total number of direct children of the class	0	Bag { }
Total number of direct ancestors of the class	0	Bag { }
Total number of interfaces the class implements	1	Bag { 'InterfazQuery' }
Total number of inherited attributes	0	
Total number of inherited operations	0	

Ilustración 29: Diseño técnico: Pestaña Class, evaluar métricas.

- **Interface:** esta pestaña permite al usuario seleccionar una interfaz determinada de entre todas las interfaces existentes en la aplicación Java que se está evaluando.

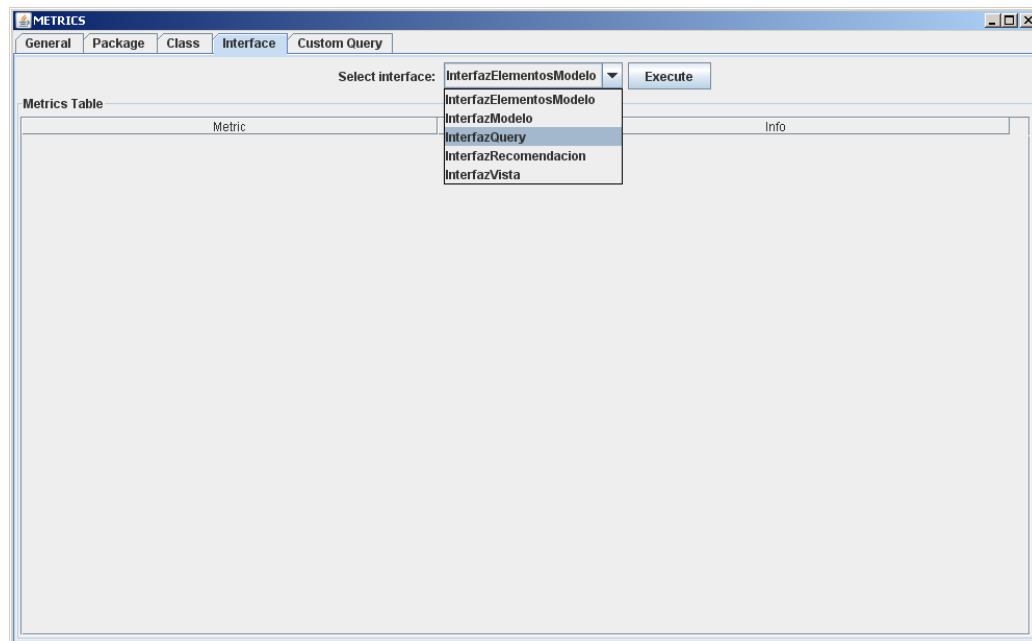


Ilustración 30: Diseño técnico: Pestaña Interface, seleccionar una interfaz.

Una vez que el usuario selecciona una determinada interfaz y pulsa el botón “Execute” se muestra una tabla con toda la información resultante al evaluar las métricas correspondientes al grupo “interface” sobre la clase que ha seleccionado el usuario.

Metric	Result	Info
Total number of Methods in the interface	4	Bag ('agregarElementoModelo', 'eliminarElementoModelo', 'recuperarElementoModelo', 'existeElementoModelo')
Total number of class directly implements this interface	0	Bag ()
The number of times the interface is externally used as attribute type	0	There is no Info for this metric.
The number of times the interface is externally used as parameter type	0	There is no Info for this metric.
The number of parameters in the interface having another interface as their type	3	Bag ('e', 'e', 'e')
The number of parameters in the interface having a class as their type	0	Bag ()

Ilustración 31: Diseño técnico: Pestaña Interface, evaluar métricas.

- **Custom Query:** esta pestaña permite al usuario seleccionar cualquier métrica especificada en el sistema independientemente del grupo al que pertenezca.

Si desea ejecutar una métrica concreta del grupo de métricas generales, basta con seleccionar la opción “*General*” en el apartado “*Check the type of metric*” y seleccionar la métrica deseada en el desplegable “*Select metric to Execute*”.

La siguiente ilustración muestra un ejemplo concreto: seleccionar la métrica “*Total number of methods in the Project*” del grupo de métricas generales:

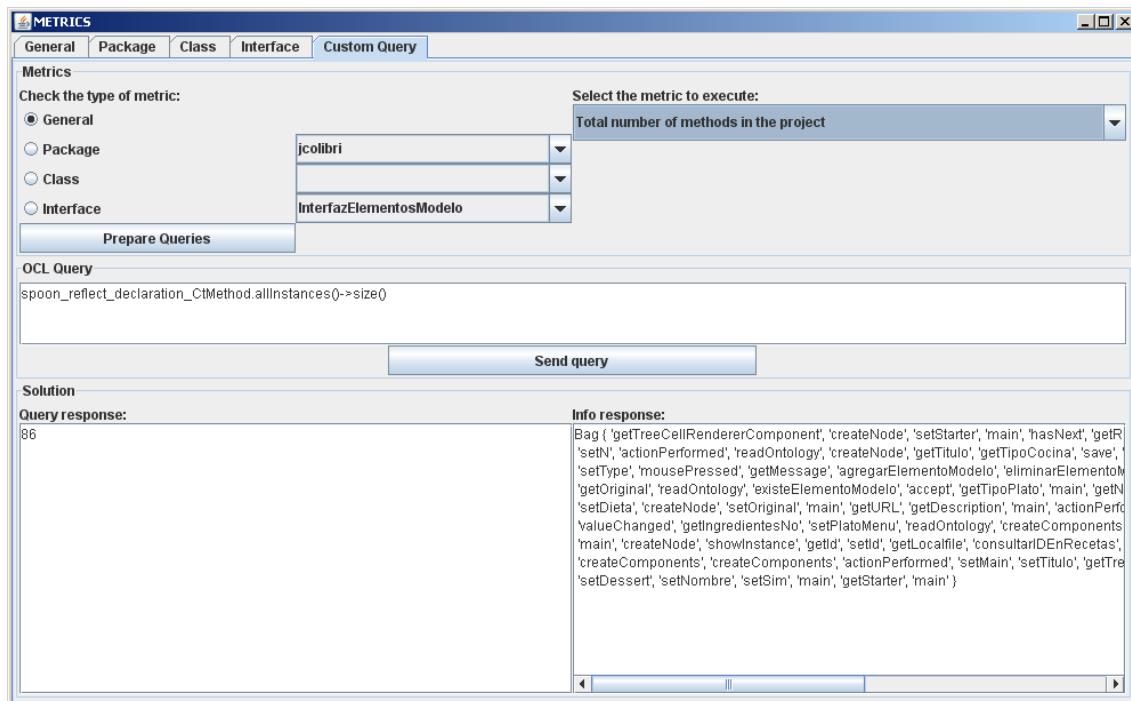


Ilustración 32: Diseño técnico: Pestaña Custom Query, evaluar métrica general.

Si desea ejecutar una métrica concreta del grupo de métricas por paquete, deberá realizar las siguientes acciones:

1. Seleccionar la opción “*Package*” en el apartado “*Check the type of metric*”.
2. Seleccionar el paquete sobre el que desea ejecutar la consulta.
3. Pulsar el botón “*Prepare queries*”.
4. Y seleccionar la métrica deseada del desplegable “*Select metric to Execute*”.

La siguiente ilustración muestra un ejemplo concreto: seleccionar el paquete “*exceptions*” y evaluar la métrica “*Number of classes in a package*”.

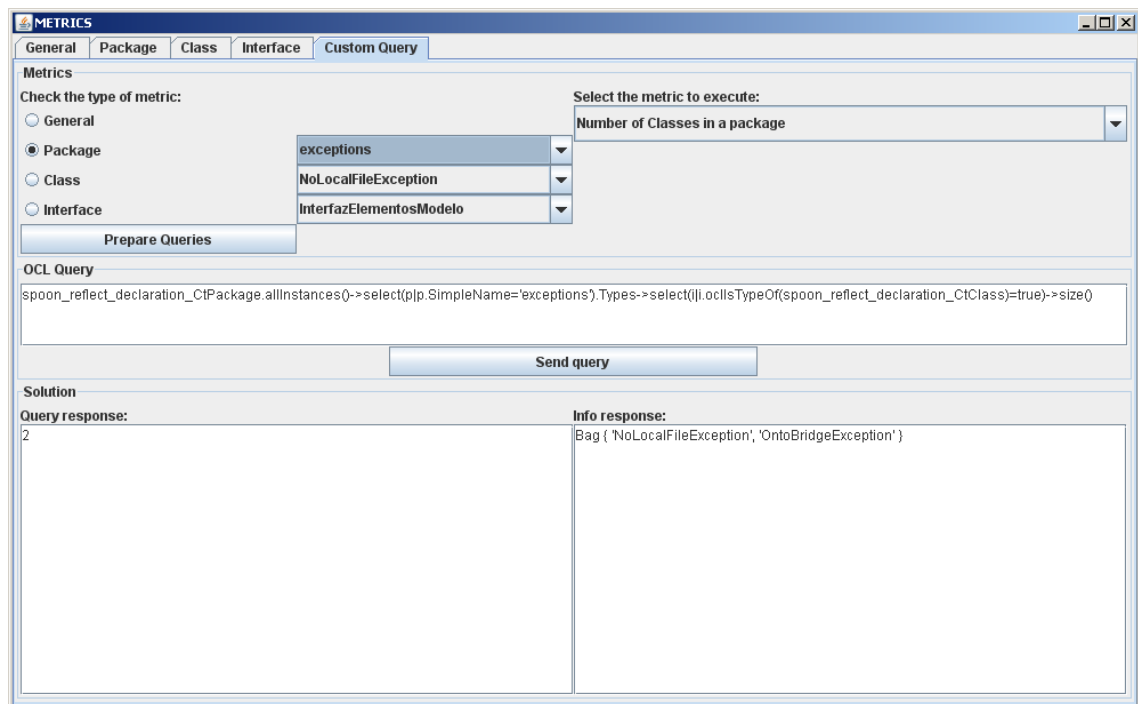


Ilustración 33: Diseño técnico: Pestaña Custom Query, evaluar métrica por paquete.

Si desea ejecutar una métrica concreta del grupo de métricas por clase, deberá realizar las siguientes acciones:

1. Seleccionar la opción “Class” en el apartado “Check the type of metric”.
2. Seleccionar la clase sobre el que desea ejecutar la consulta.
3. Pulsar el botón “Prepare queries”.
4. Y seleccionar la métrica deseada del desplegable “Select metric to Execute”.

La siguiente ilustración muestra un ejemplo concreto: seleccionar la clase “PlatoRecomendado” y evaluar la métrica “Total number of interfaces the class implements”.

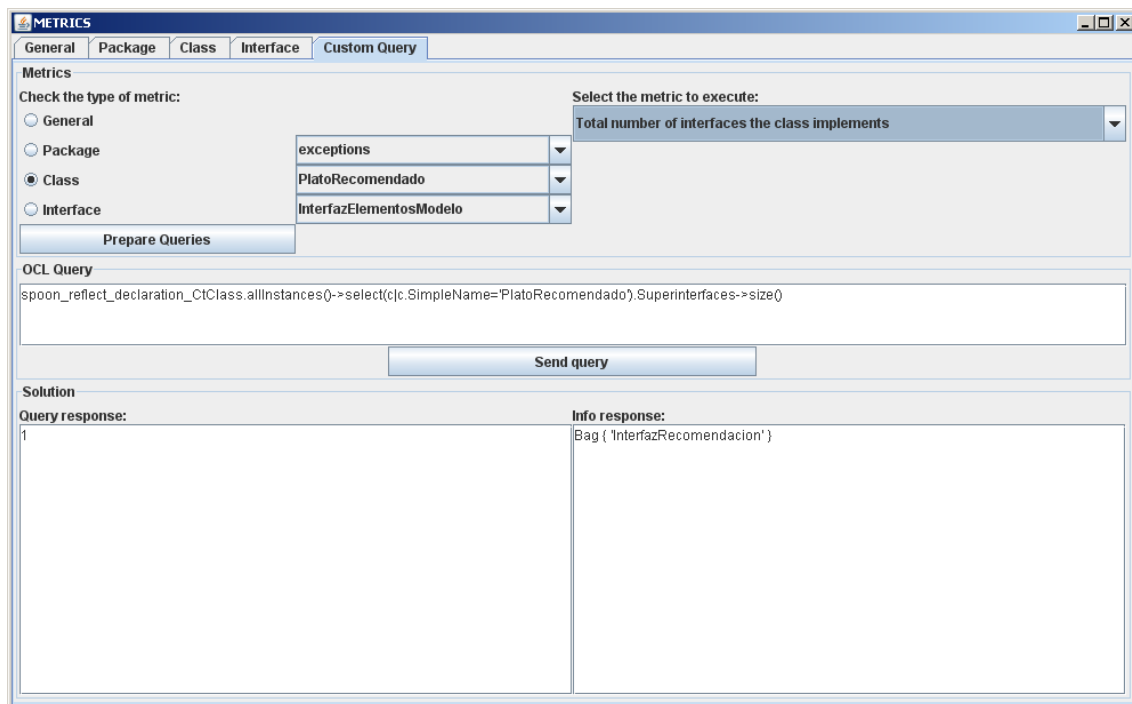


Ilustración 34: Diseño técnico: Pestaña Custom Query, evaluar métrica por clase.

Si desea ejecutar una métrica concreta del grupo de métricas por interfaz, deberá realizar las siguientes acciones:

1. Seleccionar la opción “*Interface*” en el apartado “*Check the type of metric*”.
2. Seleccionar la interfaz sobre el que desea ejecutar la consulta.
3. Pulsar el botón “*Prepare queries*”.
4. Y seleccionar la métrica deseada del desplegable “*Select metric to Execute*”.

La siguiente ilustración muestra un ejemplo concreto: seleccionar la interfaz “*InterfazModelo*” y evaluar la métrica “*The number of parameters in the interface having another interface as their type*”:

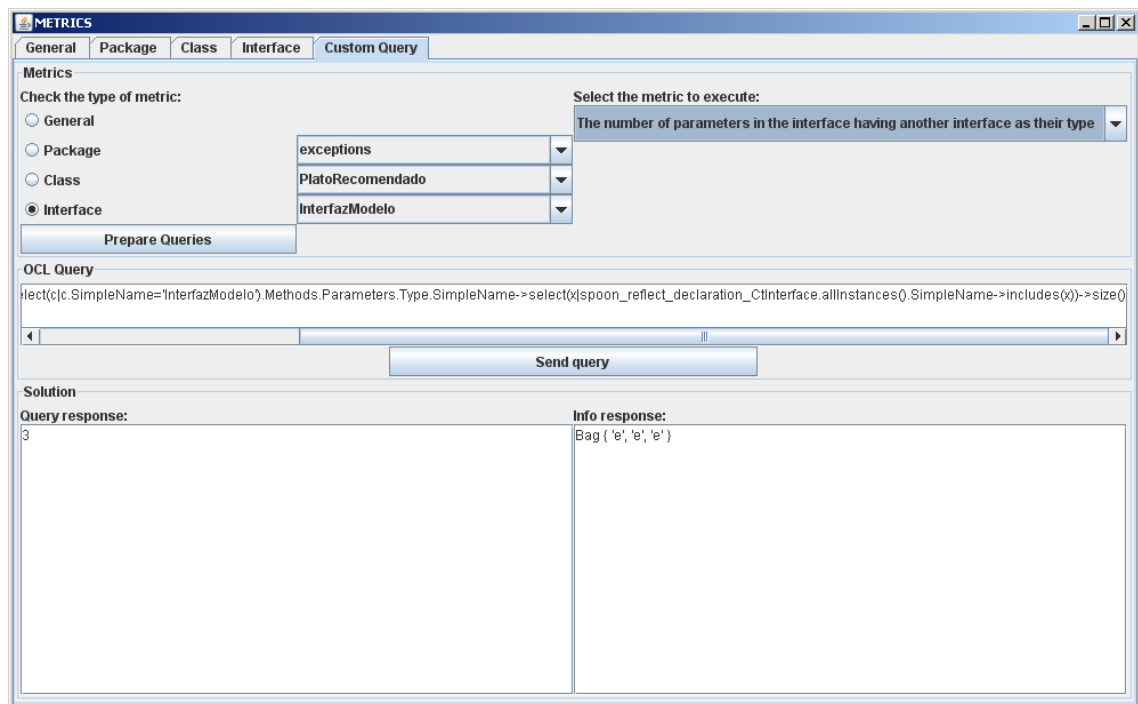


Ilustración 35: Diseño técnico: Pestaña Custom Query, evaluar métrica por interfaz.

Además, la pestaña Custom Query permite al usuario introducir nuevas consultas que no se han especificado en el sistema; para ello deberá introducir la expresión OCL correspondiente en el cuadro de texto “OCL Query” y pulsar el botón “Send Query”, el resultado de la consulta introducida se mostrará en el apartado “Query Response”.

4. Resultados Obtenidos

En este apartado se muestran los resultados obtenidos al ejecutar la herramienta **spoonToEOS** sobre algunos proyectos Java seleccionados para realizar algunas pruebas. Además, se incluye una comparativa entre spoonToEOS y el resto de herramientas.

4.1 Pruebas de ejecución

A continuación, se muestran algunos de los proyectos utilizados en las pruebas de ejecución de spoonToEOS:

- *ProjectForTest*: proyecto de prueba inicial que se ha ido utilizando para probar cada una de las métricas definidas en el sistema.
- *Motor*: proyecto Java implementado en la asignatura de Ingeniería del Software que implementa un juego para dispositivos móviles por bluetooth.
- *JadaCook_v2*: proyecto Java implementado en la asignatura Aprendizaje Automático impartida en el Máster de Investigación en Informática que implementa un recomendador CBR que aplica estrategias de Data Mining para recomendar recetas de cocina aprendiendo en función de selecciones previas de usuarios anteriores de la aplicación.

A continuación se muestran algunos ejemplos concretos que se han obtenido al evaluar cada uno de los programas anteriores:

ProjectForTest

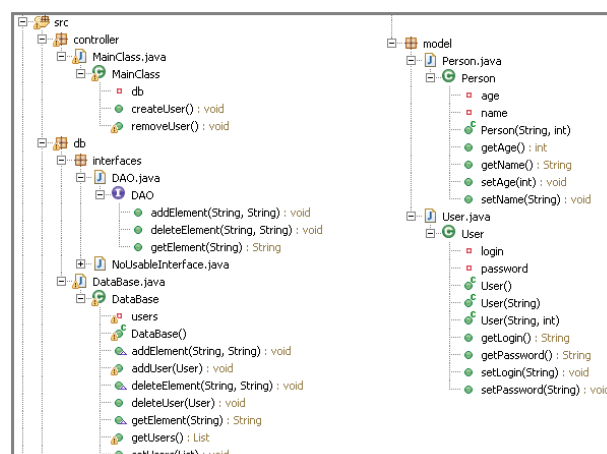


Ilustración 36: Resultados Obtenidos: Estructura ProjectForTest.

La siguiente ilustración muestra el resultado de las métricas generales del proyecto, se puede apreciar el número y nombre de los paquetes, clases, atributos y métodos que contiene el programa, así como la herencia que existe entre la clase *User* y *Person*.

METRICS		
General Package Class Interface Custom Query		
Metrics Table		
Metric	Result	Info
Total number of packages in the project	4	Bag { 'controller', 'model', 'interfaces', 'db' }
Total number of classes in the project	4	Bag { 'Person', 'DataBase', 'MainClass', 'User' }
Total number of attributes in the project	6	Bag { 'name', 'db', 'users', 'password', 'login', 'age' }
Total number of methods in the project	20	Bag { 'setLogin', 'setAge', 'getPassword', 'getUsers', 'setPassword', 'setUsers', 'getName', 'addElement', 'setName', 'deleteElement', 'addUser', 'getElement', 'getLogin', 'addElement', 'getAge', 'deleteElement', 'getElement', 'deleteUser', 'createUser', 'removeUser' }
Total number of class that are children in the project	1	Bag { 'User' }
Total number of class that are parents in the project	1	Bag { 'Person' }
Total number of If expressions in the project	1	There is no Info for this metric.
Total number of For expressions in the project	1	There is no Info for this metric.
Total number of While expressions in the project	0	There is no Info for this metric.

Ilustración 37: Resultados Obtenidos: ProjectForTest - Métricas generales.

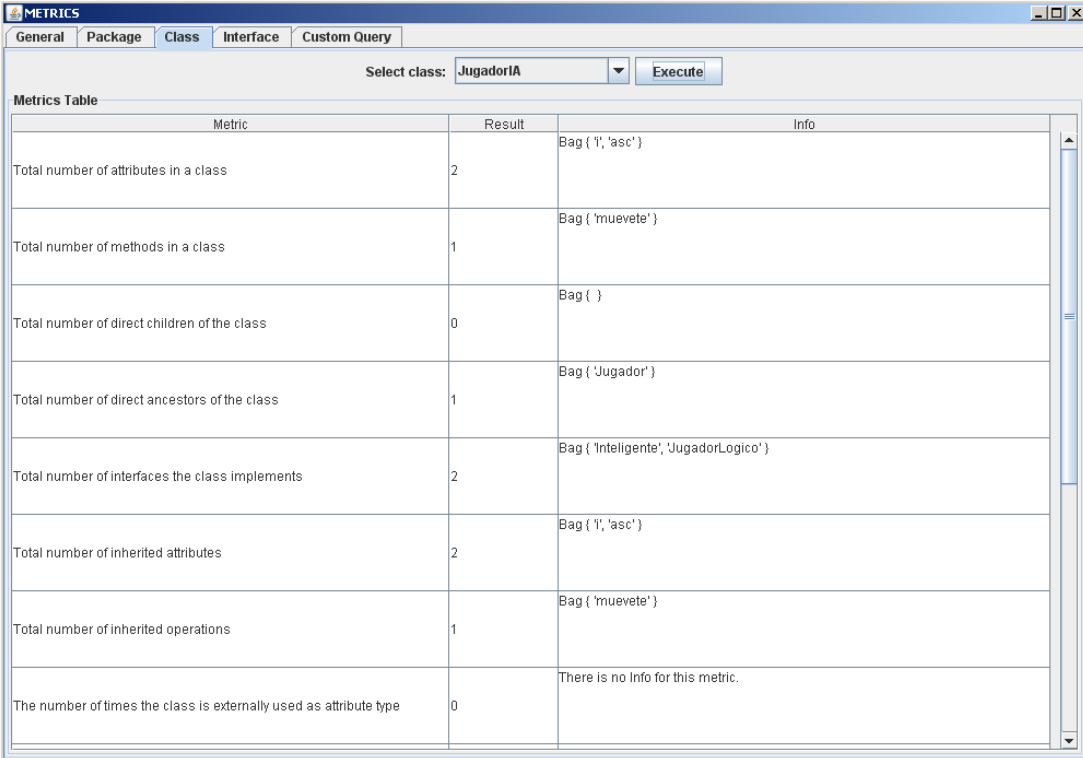
Como ejemplos concretos podemos destacar la métrica que indica que la interfaz *NoUsableInterface* no es implementada por ninguna clase del proyecto:

Total number of interfaces that are not referenced	1	Bag { 'NoUsableInterface' }
--	---	-----------------------------

Ilustración 38: Resultados Obtenidos: ProjectForTest – Interfaz no implementada.

Motor

A continuación se muestra un ejemplo de ejecución sobre este proyecto, concretamente la evaluación de una determinada clase: *JugadorIA*, clase encargada de implementar un jugador concreto, la siguiente ilustración muestra los atributos declarados en la citada clase, sus métodos, las clases de las que hereda y las interfaces que implementa.



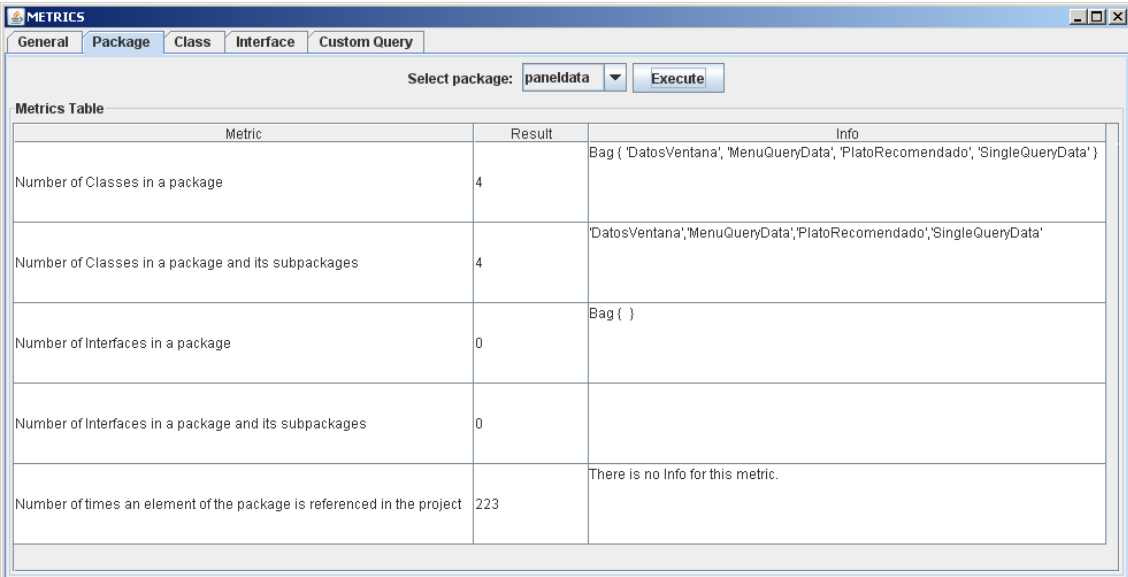
Metrics Table

Metric	Result	Info
Total number of attributes in a class	2	Bag { 'I', 'asc' }
Total number of methods in a class	1	Bag { 'muevete' }
Total number of direct children of the class	0	Bag { }
Total number of direct ancestors of the class	1	Bag { 'Jugador' }
Total number of interfaces the class implements	2	Bag { 'Inteligente', 'JugadorLogico' }
Total number of inherited attributes	2	Bag { 'I', 'asc' }
Total number of inherited operations	1	Bag { 'muevete' }
The number of times the class is externally used as attribute type	0	There is no Info for this metric.

Ilustración 39: Resultados Obtenidos: Motor – Evaluación de una clase.

JadaCook_v2

A continuación se muestra un ejemplo de ejecución sobre este proyecto, concretamente la evaluación de un determinado paquete: *paneldata*, paquete encargado de almacenar las clases que implementan la vista de la aplicación cliente. La siguiente ilustración muestra las clases que contiene, así como el número de veces que un elemento del paquete es referenciado en el proyecto.



Metrics Table

Metric	Result	Info
Number of Classes in a package	4	Bag { 'DatosVentana', 'MenuQueryData', 'PlatoRecomendado', 'SingleQueryData' }
Number of Classes in a package and its subpackages	4	'DatosVentana','MenuQueryData','PlatoRecomendado','SingleQueryData'
Number of Interfaces in a package	0	Bag { }
Number of Interfaces in a package and its subpackages	0	
Number of times an element of the package is referenced in the project	223	There is no Info for this metric.

Ilustración 40: Resultados Obtenidos: JavaCook_v2 – Evaluación de una paquete.

La siguiente tabla muestra los tiempos de ejecución de cada uno de los programas anteriores:

	Características	es.ucm.spoon.client	spoonToEOS - JET	spoonToEOS - EOS
ProjectForTest	4 paquetes. 5 clases.	≈ 5 segundos en generar un fichero XMI de 772 líneas.	≈ 10 segundos en procesar las 772 líneas del metamodelo y generar el código asociado.	< 1 segundo en ejecutar las 22 métricas iniciales.
Motor	16 paquetes. 73 ficheros.	≈ 10 segundos en generar un fichero XMI de 11258 líneas.	≈ 40 segundos en procesar las 11258 líneas del metamodelo y generar el código asociado.	< 1 segundo en ejecutar las 22 métricas iniciales.
JaDaCook_v2	18 paquetes. 40 ficheros. Librerías externas.	≈ 20 segundos en generar un fichero XMI de 15023 líneas.	≈ 2 minutos en procesar las 15023 líneas del metamodelo y generar el código asociado.	< 1 segundo en ejecutar las 22 métricas iniciales.

Tabla 5: Resultados Obtenidos: Tiempos de respuesta.

4.2 Comparativa entre herramientas

Una vez mostrados algunos ejemplos de ejecución de spoonToEOS, se incluye una comparativa entre la herramienta implementada y el resto de herramientas especificadas en el capítulo [Estado del arte](#) en función de las funcionalidades que ofrece cada una de ellas.

	JCSC	CheckStyle	JavaNCSS	JMT	Metrics 1.3.6	RSM	SDMetrics	SONAR	SpoonToEOS
Reglas de verificación de código	✓	✓	✗	✗	✗	✗	✓	✓	✗
Número de métricas considerable	✗	✗	✗	✓	✓	✓	✓	✓	✓
Límite mínimo y máximo por métrica	✓	✓	✗	✗	✓	✓	✗	✗	✗
Facilidad de añadir nuevas métricas	✗	✗	✗	✗	✗	✗	✓	✓	✓
Interfaz usable	✓	✗	✓	✗	✓	✓	✓	✓	✓
Interpretación de resultados	✓	✗	✗	✗	✓	✓	✓	✓	✗
Varios lenguajes de programación	✗	✗	✗	✗	✗	✓	✗	✓	✗
Métricas por paquete	✗	✗	✓	✗	✓	✓	✓	✗	✓
Métricas por clase	✗	✓	✓	✓	✓	✓	✓	✗	✓
Métricas por interfaz	✗	✗	✗	✗	✓	✓	✓	✗	✓
Métricas por método	✓	✗	✓	✗	✗	✓	✓	✗	✗
Métricas sobre todo el proyecto	✗	✗	✓	✓	✓	✓	✓	✓	✓

Tabla 6: Resultados Obtenidos: Comparativa entre herramientas.

Entre las funcionalidades de **spoonToEOS** cabe destacar que basta con indicar la ruta donde está almacenado el proyecto Java para el procesamiento del sistema, a diferencia de otras herramientas que necesitan que el usuario indique cada una de las clases que desea evaluar.

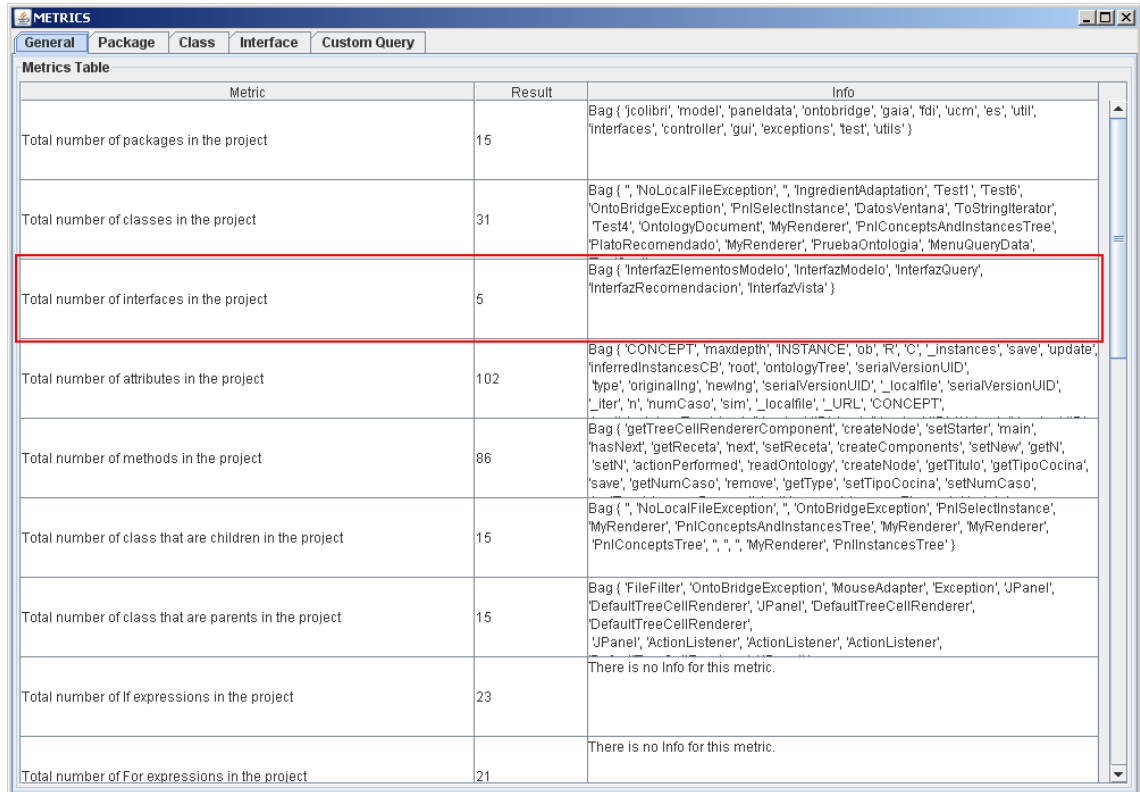
Otra de las funcionalidades a destacar de la herramienta **spoonToEOS** es la facilidad de añadir nuevas métricas. A continuación, se muestra un ejemplo concreto de cómo añadir la métrica general “*Number of interfaces in the Project*”:

1. Insertamos la nueva métrica en el fichero *metrics.xml* dentro del grupo correspondiente, en este caso dentro del grupo “*general*”. Para ello, se inserta un nuevo elemento de tipo “*metric*” con el título deseado y la expresión OCL correspondiente:

```
<group id="general">
  <metric title="Total number of packages in the project">
    <operation type="ocl">spoon_reflect_declaration_CtPackage.allInstances()->size()</operation>
    <info>spoon_reflect_declaration_CtPackage.allInstances().SimpleName</info>
  </metric>
  <metric title="Total number of classes in the project">
    <operation type="ocl">spoon_reflect_declaration_CtClass.allInstances()->size()</operation>
    <info>spoon_reflect_declaration_CtClass.allInstances().SimpleName</info>
  </metric>
  <metric title="Total number of interfaces in the project">
    <operation type="ocl">spoon_reflect_declaration_CtInterface.allInstances()->size()</operation>
    <info>spoon_reflect_declaration_CtInterface.allInstances().SimpleName</info>
  </metric>
  <metric title="Total number of attributes in the project">
    <operation type="ocl">spoon_reflect_declaration_CtField.allInstances()->size()</operation>
    <info>spoon_reflect_declaration_CtField.allInstances().SimpleName</info>
  </metric>
</group>
```

Ilustración 41: Resultados Obtenidos: Insertar nueva métrica.

2. Generamos el código de la aplicación ejecutando JET.
3. Lanzamos la aplicación y observamos que la nueva métrica aparece en la pestaña correspondiente:



Metric	Result	Info
Total number of packages in the project	15	Bag { 'jcolibri', 'model', 'paneldata', 'ontobridge', 'gaia', 'fdi', 'ucm', 'es', 'util', 'interfaces', 'controller', 'gui', 'exceptions', 'test', 'utils' }
Total number of classes in the project	31	Bag { 'NoLocalFileException', 'IngredientAdaptation', 'Test1', 'Test6', 'OntoBridgeException', 'PnlSelectInstance', 'DatosVentana', 'ToStringIterator', 'Test4', 'OntologyDocument', 'MyRenderer', 'PnlConceptsAndInstancesTree', 'PlatoRecomendado', 'MyRenderer', 'PruebaOntologia', 'MenuQueryData', 'InterfazElementosModelo', 'InterfazModelo', 'InterfazQuery', 'InterfazRecomendacion', 'InterfazVista' }
Total number of interfaces in the project	5	Bag { 'InterfazElementosModelo', 'InterfazModelo', 'InterfazQuery', 'InterfazRecomendacion', 'InterfazVista' }
Total number of attributes in the project	102	Bag { 'CONCEPT', 'maxdepth', 'INSTANCE', 'ob', 'R', 'C', '_instances', 'save', 'update', 'InferredInstancesCB', 'root', 'ontologyTree', 'serialVersionUID', 'type', 'originaling', 'newing', 'serialVersionUID', '_localfile', 'serialVersionUID', '_iter', 'n', 'numCaso', 'sim', '_localfile', '_URL', 'CONCEPT', 'maxdepth', 'INSTANCE', 'ob', 'R', 'C', '_instances', 'save', 'update', 'InferredInstancesCB', 'root', 'ontologyTree', 'serialVersionUID', 'type', 'originaling', 'newing', 'serialVersionUID', '_localfile', 'serialVersionUID', '_iter', 'n', 'numCaso', 'sim', '_localfile', '_URL', 'CONCEPT' }
Total number of methods in the project	86	Bag { 'getTreeCellRendererComponent', 'createNode', 'setStarter', 'main', 'hasNext', 'getReceta', 'next', 'setReceta', 'createComponents', 'setNew', 'getN', 'setN', 'actionPerformed', 'readOntology', 'createNode', 'getTitulo', 'getTipoCocina', 'save', 'getNumCaso', 'remove', 'getType', 'setTipoCocina', 'setNumCaso', 'getTreeCellRendererComponent', 'createNode', 'setStarter', 'main', 'hasNext', 'getReceta', 'next', 'setReceta', 'createComponents', 'setNew', 'getN', 'setN', 'actionPerformed', 'readOntology', 'createNode', 'getTitulo', 'getTipoCocina', 'save', 'getNumCaso', 'remove', 'getType', 'setTipoCocina', 'setNumCaso' }
Total number of class that are children in the project	15	Bag { 'NoLocalFileException', 'OntoBridgeException', 'PnlSelectInstance', 'MyRenderer', 'PnlConceptsAndInstancesTree', 'MyRenderer', 'MyRenderer', 'PnlConceptsTree', 'MyRenderer', 'PnlInstancesTree' }
Total number of class that are parents in the project	15	Bag { 'FileFilter', 'OntoBridgeException', 'MouseAdapter', 'Exception', 'JPanel', 'DefaultTreeCellRenderer', 'JPanel', 'DefaultTreeCellRenderer', 'DefaultTreeCellRenderer', 'JPanel', 'ActionListener', 'ActionListener', 'ActionListener' }
Total number of If expressions in the project	23	There is no Info for this metric.
Total number of For expressions in the project	21	There is no Info for this metric.

Ilustración 42: Resultados Obtenidos: Nueva métrica disponible.

5. Conclusiones

Este apartado resume las conclusiones que se pueden obtener una vez concluido el trabajo realizado para el proyecto de Máster en Investigación en Informática.

Al finalizar el proyecto, podemos echar la vista atrás y hacer un balance de los objetivos cumplidos a partir de los objetivos propuestos al inicio del proyecto; así como los nuevos objetivos que han ido surgiendo en las distintas fases por las que ha pasado el proyecto para solventar los distintos problemas que nos hemos ido encontrando.

Por otro lado, tenemos que tener en cuenta que este proyecto, y en concreto la herramienta spoonToEOS puede ofrecer mucho más, es decir, existen numerosos caminos por los que podemos seguir investigando para conseguir una herramienta mucho más robusta y capaz de ofrecer una evaluación más detallada de cualquier software.

5.1 Objetivos Cumplidos

Como objetivos iniciales del proyecto se propusieron los siguientes:

- Investigación de métricas de calidad del software y de herramientas que evaluaran código fuente.
- Implementar una herramienta capaz de evaluar métricas sobre un determinado programa Java utilizando el componente EOS.

El primer punto, se consiguió en la primera fase del proyecto al investigar el estado del arte de esta metodología; mientras que para conseguir el segundo objetivo, se fueron estableciendo nuevos objetivos para ir consiguiendo poco a poco las distintas funcionalidades de la herramienta final:

- En primer lugar, se propuso conseguir una instancia del metamodelo de java a partir de un programa Java concreto; damos por alcanzado este objetivo una vez que aprendemos cómo SpoonEMF genera el metamodelo de un proyecto Java almacenándolo en un fichero *.xmi, lo cual implica en aprender el metamodelo de Java y el framework EMF que Eclipse utiliza para interpretar los proyectos.
- A continuación, se propone que el código de la aplicación se autogenera con JET, lo cual, implica aprender a utilizar este plugin de eclipse y a adaptar sus funcionalidades a nuestras necesidades, es decir, a conseguir generar el código fuente de la aplicación. Este objetivo concreto, ha sido uno de los objetivos más costosos de cumplir, puesto que sin conocer las capacidades de JET para autogenerar código fuente, conseguimos generar un conjunto de paquetes y clases que por un lado son capaces de interactuar con el componente EOS, insertando el metamodelo de java y la instancia del

metamodelo de Java y enviando consultas OCL, y por otro lado muestra al usuario una interfaz que le permite interactuar con la aplicación.

- Una vez conseguidos estos dos puntos anteriores, ya hemos conseguido una primera aproximación a la herramienta que buscamos, faltaría conseguir el objetivo principal del proyecto: evaluar un proyecto Java a partir de métricas de miden la calidad del software.

Este objetivo se puede resumir en agrupar todas aquellas métricas encontradas que nos interesan para nuestra herramienta, añadir métricas que nos resulten interesantes y obtener la expresión OCL correspondiente a cada una de ellas. Este objetivo, implica aprender el lenguaje OCL y ser capaces de expresar las métricas escogidas para nuestra herramienta.

En resumen, en este proyecto no sólo nos ha ayudado a conocer cómo se evalúa la calidad del software, sino que además, se han aprendido nuevos conceptos, nuevas metodologías de programación y el uso de nuevas tecnologías.

5.2 Trabajo Futuro

A partir del trabajo realizado, se pueden considerar distintos caminos que se pueden seguir para continuar y mejorar el proyecto. La herramienta spoonToEOS, en su primera versión, implementa muchas de las funcionalidades que realiza cualquier otra herramienta de evaluación, pero se puede mejorar para conseguir una herramienta más robusta y eficaz.

Se distinguen las siguientes vías a seguir para continuar con el trabajo realizado:

- Facilitar el uso de la aplicación:

Para facilitar el uso de la aplicación, se propone implementar un plugin de Eclipse que permita al usuario abrir la interfaz de spoonToEOS a partir de un determinado proyecto ya importado en el Eclipse.

- Mejorar la aplicación actual:

Para mejorar la aplicación actual, se propone dedicar un cierto tiempo a la investigación de los tiempos de respuesta y las posibles formas de mejorarlos.

Se propone además, mejorar la interfaz de usuario, ya que esta primera versión implementa una interfaz básica que permite al usuario visualizar de forma rápida la información de su proyecto.

- Añadir nueva funcionalidad:

Para añadir nueva funcionalidad a la aplicación se pueden considerar los siguientes aspectos:

- Añadir nuevas métricas con sus correspondientes expresiones OCL.

- Añadir nuevos parámetros por métrica que permita al usuario establecer el máximo y mínimo permitido, por ejemplo máximo y mínimo de métodos por clase.
- Interpretación de resultados, es decir, una vez obtenidos los resultados de cada una de las métricas ejecutadas, ofrecer al usuario una interpretación de las mismas, indicando al usuario si debe mejorar algún aspecto del código fuente.
- Contemplar el análisis del código fuente para obtener un código fácilmente interpretable, es decir, permitir al usuario definir reglas que se deben cumplir en el código fuente, por ejemplo, que los nombres de clases empiecen por mayúsculas, que los métodos tengan asociado un comentario inicial, etcétera. En general, haciendo uso de la flexibilidad del lenguaje OCL podrían evaluarse invariantes que sean de interés para el usuario sobre la sintaxis de los programas escritos como instancia del metamodelo o definirse queries complejos como devolviendo, por ejemplo, tuplas con (nombre de clase, valor de métrica sobre la clase).

Apéndice A. SpoonJDT

El plugin de Spoon para eclipse [\[http://spoon.gforge.inria.fr/Spoon/HomePage\]](http://spoon.gforge.inria.fr/Spoon/HomePage) se configura tal y como se muestra en la siguiente ilustración: a partir de las propiedades de un determinado proyecto Java se selecciona la opción Spoon y se indica el modo de trabajar de Spoon “Advanced”, “Default”, “NoCompile” o “NoGenerate”, indicando además el fichero de salida donde se almacenará el metamodelo del proyecto seleccionado.

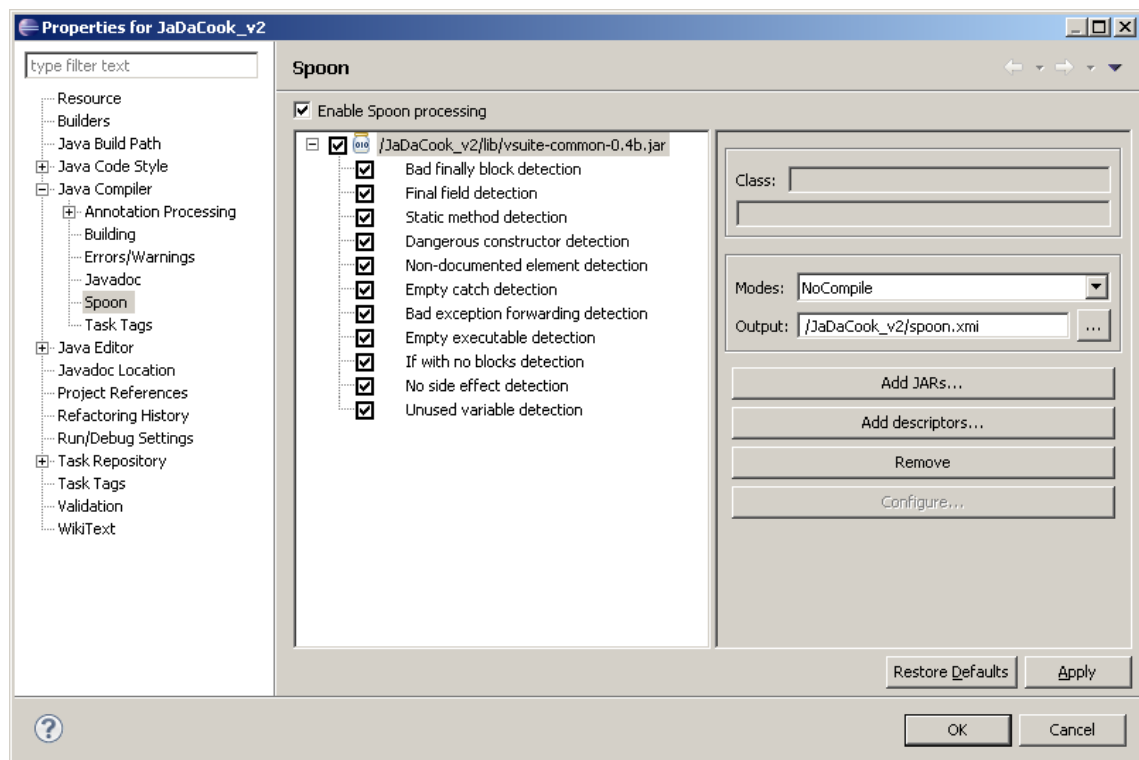


Ilustración 43: Apéndice A: Configuración SpoonJDT.

Sin embargo, este plugin presenta el siguiente problema: no genera el metamodelo de aquellos proyectos que referencian librerías que no incluyen el código fuente de sus clases. Se observa la siguiente excepción en el fichero de log del eclipse:

```
!ENTRY fr.inria.spoonEMF 4 0 2010-05-31 12:41:41.316
!MESSAGE receiverType
!STACK 0
java.lang.NoSuchFieldError: receiverType
    at spoon.support.builder.JDTTreeBuilder.visit(JDTTreeBuilder.java:2071)
    at org.eclipse.jdt.internal.compiler.ast.FieldReference.traverse(FieldReference.java:615)
    at org.eclipse.jdt.internal.compiler.ast.Assignment.traverse(Assignment.java:214)
    at spoon.support.builder.JDTTreeBuilder.visit(JDTTreeBuilder.java:2280)
    at
    org.eclipse.jdt.internal.compiler.ast.MethodDeclaration.traverse(MethodDeclaration.java:207)
    at spoon.support.builder.JDTTreeBuilder.visit(JDTTreeBuilder.java:2764)
    at org.eclipse.jdt.internal.compiler.ast.TypeDeclaration.traverse(TypeDeclaration.java:1249)
    at
    org.eclipse.jdt.internal.compiler.ast.CompilationUnitDeclaration.traverse(CompilationUnitDeclaration.java:679)
    at fr.inria.spoon.jdt.JDTCompiler.buildModel(JDTCompiler.java:61)
    at fr.inria.spoon.SpoonBuilder.process(SpoonBuilder.java:373)
    at fr.inria.spoon.SpoonBuilder.build(SpoonBuilder.java:190)
    at org.eclipse.core.internal.events.BuildManager$2.run(BuildManager.java:627)
    at org.eclipse.core.runtime.SafeRunner.run(SafeRunner.java:42)
    at org.eclipse.core.internal.events.BuildManager.basicBuild(BuildManager.java:170)
    at org.eclipse.core.internal.events.BuildManager.basicBuild(BuildManager.java:201)
    at org.eclipse.core.internal.events.BuildManager$1.run(BuildManager.java:253)
    at org.eclipse.core.runtime.SafeRunner.run(SafeRunner.java:42)
    at org.eclipse.core.internal.events.BuildManager.basicBuild(BuildManager.java:256)
    at org.eclipse.core.internal.events.BuildManager.basicBuildLoop(BuildManager.java:309)
    at org.eclipse.core.internal.events.BuildManager.build(BuildManager.java:341)
    at org.eclipse.core.internal.events.AutoBuildJob.doBuild(AutoBuildJob.java:140)
    at org.eclipse.core.internal.events.AutoBuildJob.run(AutoBuildJob.java:238)
    at org.eclipse.core.internal.jobs.Worker.run(Worker.java:55)
```

Es decir, se produce un error al encontrar un tipo de datos no incluido en el metamodelo de Java o en la instancia de metamodelo que Spoon está generando y por tanto no se genera la instancia del metamodelo de Java a partir del proyecto Java seleccionado.

Apéndice B. Cliente para SpoonEMF y limitaciones

Como el plugin de Spoon disponible para Eclipse no es útil para nuestra herramienta, puesto que sólo genera el fichero final con el metamodelo del proyecto Java seleccionado en los casos en los que no se produce ningún error, se generó un cliente de Spoon denominado **spoonClient** cuya única funcionalidad es invocar a la clase principal de SpoonEMF con los siguientes parámetros: el nombre del fichero donde se debe almacenar el metamodelo generado y la ruta donde está almacenado el proyecto Java.

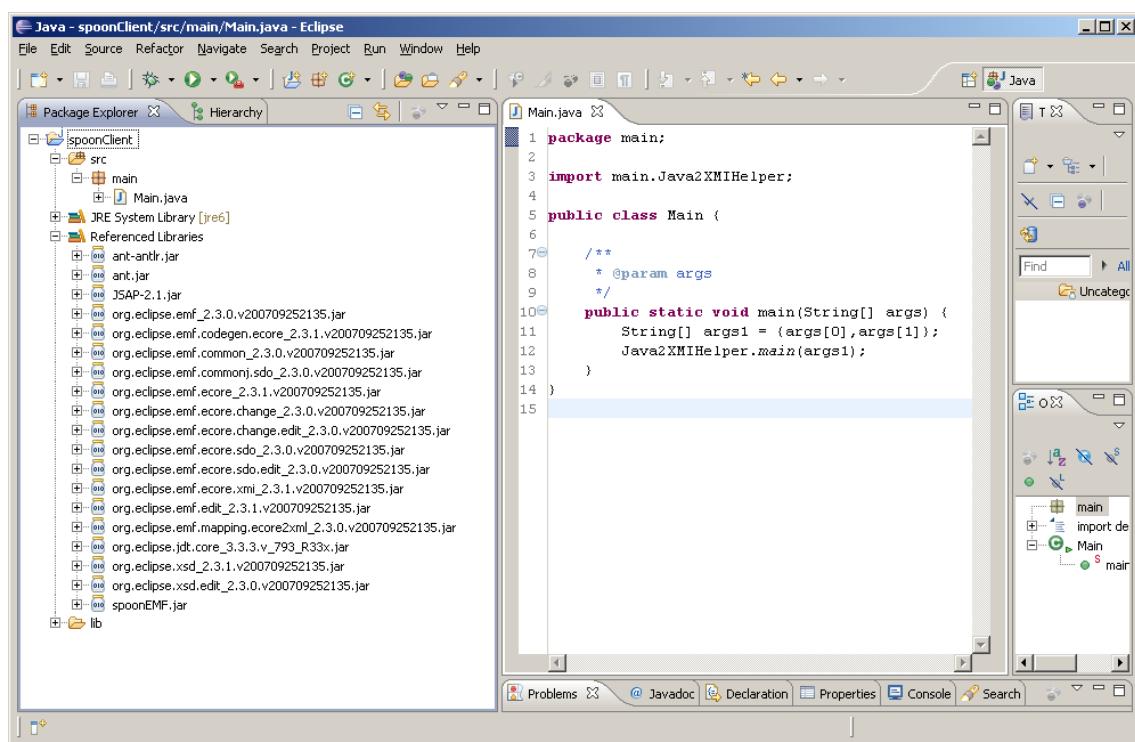


Ilustración 44: Apéndice B: Estructura spoonClient.

Para generar el metamodelo de un determinado proyecto Java basta con ejecutar el proyecto **spoonClient** tal y como se indica en la siguiente ilustración:

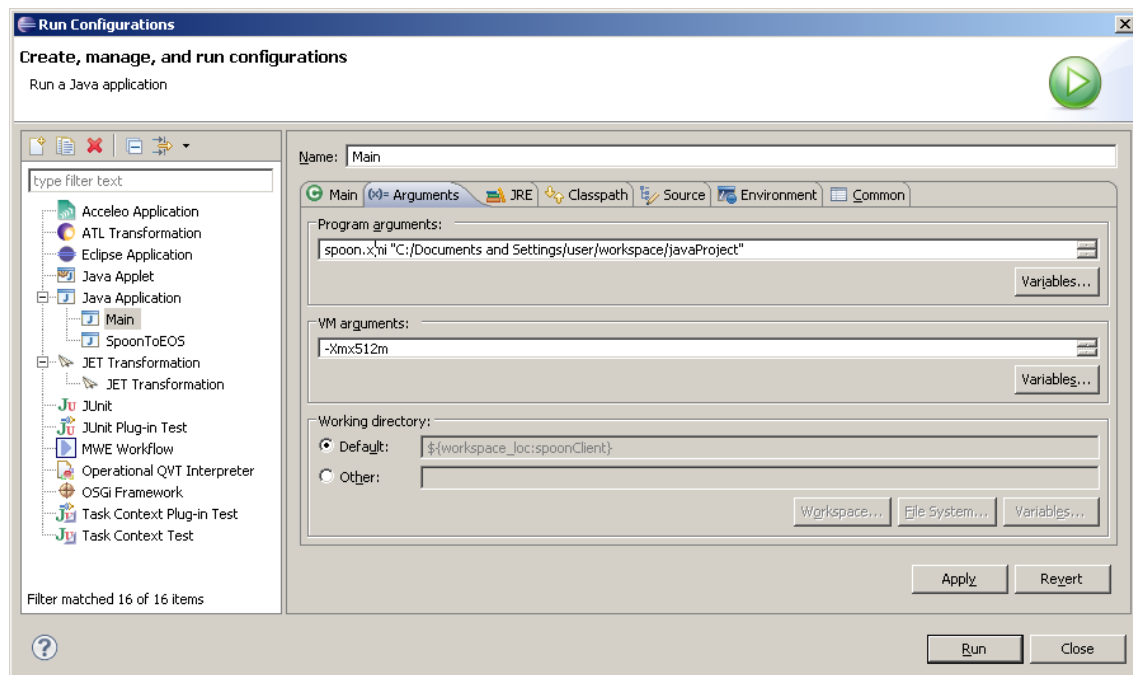


Ilustración 45: Apéndice B: Ejecución spoonClient.

Este cliente implementado para SpoonEMF tiene el mismo problema que el plugin SpoonJDT, pero con la diferencia de que siempre genera el metamodelo independientemente de que se produzcan errores al encontrar tipos de datos no reconocidos, la desventaja que asumimos es que no se incluyen las clases que referencian este tipo de datos no reconocidos procedentes de librerías que no incluyen su código fuente.

A continuación se muestran algunas pruebas de ejecución de SpoonEMF:

- Programa 1: es una aplicación Java que no incluye librerías externas, tan sólo contiene una clase principal y un paquete llamado *model* que almacena una clase llamada *User* que hereda de la clase llamada *Person*:

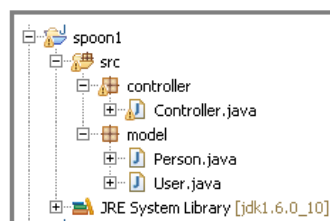


Ilustración 46: Apéndice B: Estructura Ejemplo 1.

El fichero de salida `spoon1.xmi` se genera correctamente, incluyendo todos los componentes del programa Java incluyendo la herencia entre las clases *User* y *Person*, a continuación se muestra un fragmento del fichero de salida que representa la herencia entre ambas clases:

```

<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:spoon.reflect.code="http://spoon#reflect.code"
  <spoon.reflect.declaration:CtClass Parent="/36" SimpleName="User" Fields="/164 /167" Methods="/274 /287 /318 /331" Superclass="/162" Constructors="/170"
    <Modifiers>public</Modifiers>
  </spoon.reflect.declaration:CtClass>
  <spoon.reflect.reference:CtTypeReference SimpleName="Person" Package="/163"/>
  <spoon.reflect.reference:CtPackageReference SimpleName="model"/>
  <spoon.reflect.declaration:CtField Parent="/161" SimpleName="usr" Type="/165" DeclaringType="/161">
    <Modifiers>private</Modifiers>
  </spoon.reflect.declaration:CtField>
  <spoon.reflect.reference:CtTypeReference SimpleName="String" Package="/166"/>
  <spoon.reflect.reference:CtPackageReference SimpleName="java.lang"/>
  <spoon.reflect.declaration:CtField Parent="/161" SimpleName="pass" Type="/168" DeclaringType="/161">
    <Modifiers>private</Modifiers>
  </spoon.reflect.declaration:CtField>
  <spoon.reflect.reference:CtTypeReference SimpleName="String" Package="/169"/>
  <spoon.reflect.reference:CtPackageReference SimpleName="java.lang"/>
  <spoon.reflect.declaration:CtConstructor Parent="/161" Parameters="/171 /174 /175 /178" Body="/181">
    <Modifiers>public</Modifiers>
  </spoon.reflect.declaration:CtConstructor>
  <spoon.reflect.declaration:CtConstructor>

```

Ilustración 47: Apéndice B: Resultado ejecución spoonClient Ejemplo 1.

Los identificadores de tipo “/x” hacen referencia al número de elemento XML del fichero xmi donde se define el tipo en cuestión, por ejemplo en el siguiente ejemplo, la superclase de la que hereda la clase User está definida en el 162º declaración del xmi.

- Programa 2: es una aplicación Java que contiene una clase llamada Mail con un atributo de tipo Date, por tanto incluye la librería perteneciente al JDK de Java: java.util.Date:

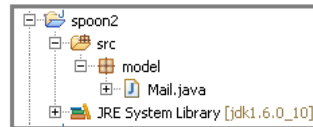


Ilustración 48: Apéndice B: Estructura Ejemplo 2.

El fichero de salida spoon2.xmi se genera correctamente, incluyendo el atributo cuyo tipo pertenece a otra librería. A continuación se muestra el fragmento de xmi que lo representa.

```

<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:spoon.reflect.code="http://spoon#reflect.code"
  <spoon.reflect.declaration:CtClass Parent="/1" SimpleName="Mail" Fields="/2" Constructors="/5">
    <Modifiers>public</Modifiers>
  </spoon.reflect.declaration:CtClass>
  <spoon.reflect.declaration:CtPackage SimpleName="model" Types="/0"/>
  <spoon.reflect.declaration:CtField Parent="/0" SimpleName="sent" Type="/3" DeclaringType="/0"/>
  <spoon.reflect.reference:CtTypeReference SimpleName="Date" Package="/4"/>
  <spoon.reflect.reference:CtPackageReference SimpleName="java.util"/>
  <spoon.reflect.declaration:CtConstructor Parent="/0" Parameters="/6" Body="/9">
    <Modifiers>public</Modifiers>
  </spoon.reflect.declaration:CtConstructor>

```

Ilustración 49: Apéndice B: Resultado ejecución spoonClient Ejemplo 2.

- Programa 3: es una aplicación Java que incluye una librería externa qtjambi-4-4.3_01.jar y contiene una clase denominada External con un atributo de tipo QWidget, clase perteneciente a dicha librería externa:

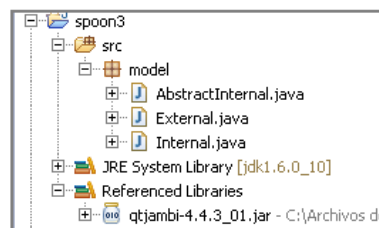


Ilustración 50: Apéndice B: Estructura Ejemplo 3.

El fichero de salida spoon3.xmi no se genera correctamente y SpoonEMF genera los siguientes errores:

```

<terminated> spoonClient [Java Application] C:\Archivos de programa\Java\jdk1.6.0_10\bin\javaw.exe (26/07/2009 20:32:55)
  folder : [C:\Documents and Settings\Ana\workspace\spoon3]
  beginning to parse[C:\Documents and Settings\Ana\workspace\spoon3]
  Filename : C:\Documents and Settings\Ana\workspace\spoon3\src\model\External.java
  COMPILED type(s)
  5 PROBLEM(s) detected
    - Pb(390) The import com.trolltech cannot be resolved
    - Pb(2) QWidget cannot be resolved to a type
    - Pb(50) widget cannot be resolved
    - Pb(50) widget cannot be resolved
    - Pb(2) QWidget cannot be resolved to a type
  end
  Number of elements after parsing the folder 63
  beginning saving
  end saving
  
```

Ilustración 51: Apéndice B: Errores ejecución spoonClient Ejemplo 3.

El fichero de salida no incluye la clase que importa la librería externa, en este caso no incluye la descripción del código correspondiente a la clase External.

El fichero de salida tan sólo hace referencia a la clase External cuando otra clase incluye un atributo de tipo External u otra clase extiende de External. En este caso el fichero de salida hace referencia a un atributo de tipo External pero no incluye el código de dicha clase.

```

<spoon.reflect.declaration:CtClass Parent="/1" SimpleName="Internal" Fields="/12" Superclass="/10" Constructors="/15">
  <Modifiers>public</Modifiers>
</spoon.reflect.declaration:CtClass>
<spoon.reflect.reference:CtTypeReference SimpleName="AbstractInternal" Package="/11"/>
<spoon.reflect.reference:CtPackageReference SimpleName="model"/>
<spoon.reflect.declaration:CtField Parent="/9" SimpleName="external" Type="/13" DeclaringType="/9"/>
<spoon.reflect.reference:CtTypeReference SimpleName="External" Package="/14"/>
<spoon.reflect.reference:CtPackageReference SimpleName="model"/>
<spoon.reflect.declaration:CtConstructor Parent="/9" Body="/16">
  <Modifiers>public</Modifiers>
</spoon.reflect.declaration:CtConstructor>
  
```

Hace referencia al atributo de tipo External pero no hay un CtClass que identifique la clase External que contiene librerías external al jdk de java.

Ilustración 52: Apéndice B: Resultado ejecución spoonClient Ejemplo 3.

- Programa 4: es una aplicación Java que incluye librerías externas que sí incluyen su código fuente.

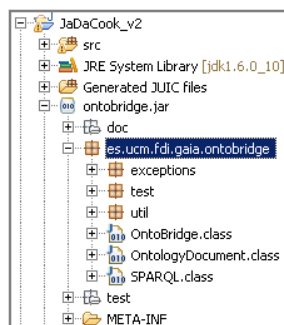


Ilustración 53: Apéndice B: Estructura Ejemplo 4.

Como resultado se obtiene un fichero de salida que representa en el metamodelo tanto las clases y paquetes de la librería referenciada como las clases y paquetes que las referencian desde el proyecto Java seleccionado.

```
< spoon.reflect.declaration:CtPackage SimpleName="es" Packages="/2"/>
< spoon.reflect.declaration:CtPackage Parent="/1" SimpleName="ucm" Packages="/3" DeclaringPackage="/1"/>
< spoon.reflect.declaration:CtPackage Parent="/2" SimpleName="fdi" Packages="/4" DeclaringPackage="/2"/>
< spoon.reflect.declaration:CtPackage Parent="/3" SimpleName="gaia" Packages="/5" DeclaringPackage="/3"/>
< spoon.reflect.declaration:CtPackage Parent="/4" SimpleName="ontobridge" Packages="/174 /249 /10966" Types="/0" DeclaringPackage="/4"/>
< spoon.reflect.declaration:CtField Parent="/0" SimpleName="_URL" Type="/7" DeclaringType="/0">
  <Modifiers>private</Modifiers>
</ spoon.reflect.declaration:CtField>
```

Ilustración 54: Apéndice B: Resultado ejecución spoonClient Ejemplo 4.

A partir de este código se ha implementado el plugin `es.ucm.spoon.client` explicado en el apartado [Módulo `es.ucm.spoon.client`](#) del presente documento.

Apéndice C. Plantilla inicial main.jet

Este apartado simplemente recorre la plantilla inicial main.jet que se utiliza en el componente spoonToEOS para comenzar la generación del código fuente de la aplicación resultante.

```
<%@taglib prefix="ws" id="org.eclipse.jet.workspaceTags" %>

<c:setVariable var="org.eclipse.jet.taglib.control.iterateSetsContext" select="true()" />

<%@jet imports="org.eclipse.emf.ecore.*"%>

<ws:project name="spoonToEOS">
  <ws:folder path="src">
    <java:package name="es.ucm.main">
      <java:class name="MainClass" template="templates/main_process.java.jet"/>
    </java:package>
    <java:package name="es.ucm.model">
      <java:class name="ClassDiagramToEOS" template="templates/ecore_process.java.jet"/>
      <java:class name="ObjectDiagramToEOS" template="templates/spoon_report.java.jet"/>
    </java:package>
    <java:package name="es.ucm.view">
      <java:class name="OpenProject" template="templates/open_project.java.jet"/>
      <java:class name="View" template="templates/gui.java.jet"/>
      <java:class name="Query" template="templates/query.java.jet"/>
    </java:package>
    <java:package name="es.ucm.view.panel">
      <java:class name="MetricsPanel" template="templates/metrics_panel.java.jet"/>
      <java:class name="SpecificPanel" template="templates/specific_panel.java.jet"/>
      <java:class name="GeneralPanel" template="templates/general_panel.java.jet"/>
      <java:class name="CustomPanel" template="templates/custom_panel.java.jet"/>
    </java:package>
    <java:package name="es.ucm.util">
      <java:class name="ComplexQueries" template="templates/complex_queries.java.jet"/>
      <java:class name="TextAreaRenderer"
        template="templates/textarea_renderer.java.jet"/>
    </java:package>
  </ws:folder>
</ws:project>
```


Apéndice D. Instalación y ejecución

Para ejecutar la primera versión de la herramienta **spoonToEOS** se deberán seguir los siguientes pasos:

1. Instalar Eclipse Modeling Tools [<http://www.eclipse.org/downloads/>].
2. Copiar en la carpeta plugins del Eclipse recién instalado los siguientes plugins:
 - a. es.ucm.spoon.client_1.0.0.201006011823.jar
 - b. SpoonEMF2EclipsePlugin
[<http://soft.vub.ac.be/soft/research/mdd/spoonemf2>]
3. Ejecutar Eclipse y verificar el aparece el menú “SpoonEMF” en la barra de menú de Eclipse.
4. Seleccionar la opción “Generate XMI” del menú “SpoonEMF” e introducir la ruta donde está almacenado el proyecto Java.
5. En este punto ya hemos ejecutado la primera fase: obtener el metamodelo del proyecto Java.
6. Para ejecutar spoonToEOS se deberá importar el proyecto spoonToEOS al Eclipse.
7. Se deberá copiar el fichero obtenido en el punto 4 en la carpeta models del proyecto spoonToEOS.
8. Se deberá ejecutar el proyecto spoonToEOS con la opción JET Transformation tal y como se indica en la siguiente ilustración:

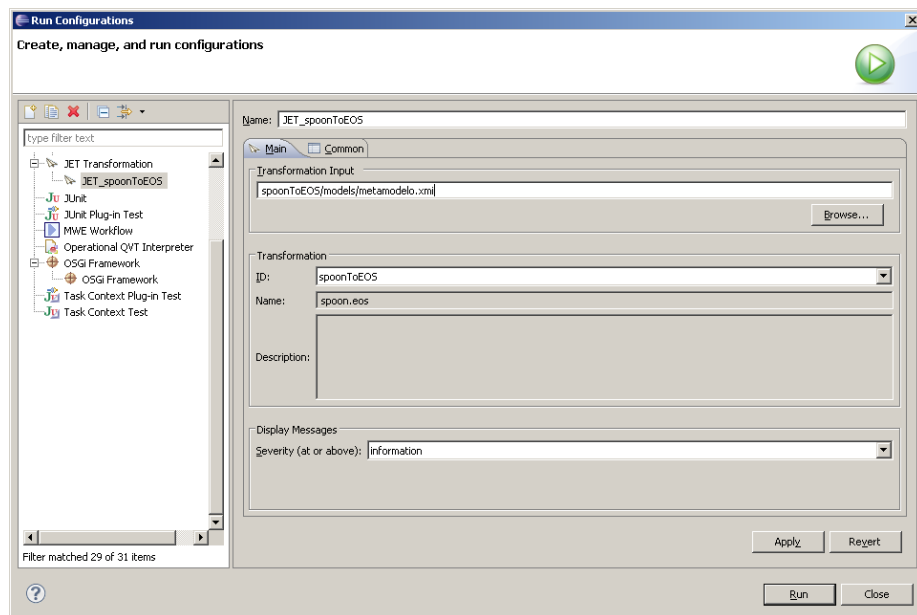


Ilustración 55: Apéndice D: Generación código fuente spoonToEOS.

9. Una vez generado el código fuente de la aplicación en la carpeta src se podrá ejecutar la aplicación como un aplicación Java tal y como indica la siguiente ilustración:

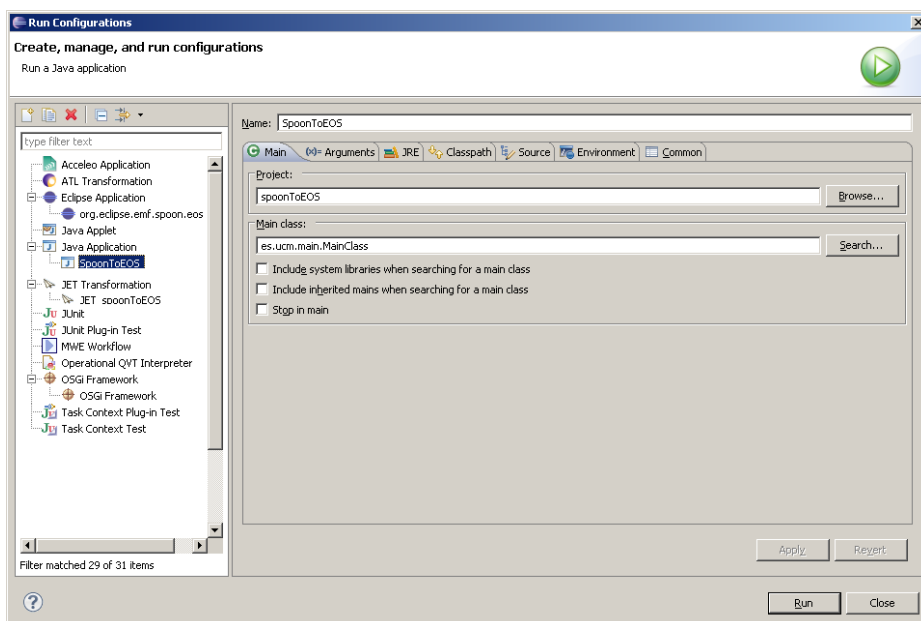


Ilustración 56: Apéndice D: Ejecución spoonToEOS.

10. Finalmente al ejecutar la clase principal del código generado en el punto 8 se abrirá la interfaz de usuario de spoonToEOS.

Bibliografía

Libros y artículos:

- [1] OMG Object Management Group: *Metamodel and UML Profile for Java and EJB Specification*. Febrero 2004, Version 1.0. Página 14.
- [2] M. Monperrus, J-M Jézéquel, J. Champeau, and B. Hoeltzener. A Model-driven Measurement Approach. ENSIETA - Brest (Fr). INRIA & University of Rennes (Fr).
- [3] M. Clavel, M. Egea, M. A. García de Dios. Building an Efficient Component for OCL Evaluation.
- [4] OMG Object Management Group: Object Constraint Language OMG Available Specification Version 2.0. 2001. Capítulo 7.
- [5] Object Management Group. Object constraint language specification (2.0). Technical report, Frammingam, Mass, 2006. <http://www.omg.org>.
- [6] Métricas para Sistemas Orientados a Objetos.
- [7] A.J. Fuente, R. Izquierdo, J.M Cueva, B. López, L. Joyanes. Sistema de Métricas para Tiempo Real en Aplicaciones Java.
- [8] Norman E. Fenton, Thonson Computer Press. Software Metrics: A rigorous approach. ISBN: 1-85032-242-2 (1991)
- [9] Medición para la gestión en Ingeniería del Software Dolado, J. et al. 2000, Ra-ma ISBN: 84-7897-403-2.
- [10] Y. Crespo, C. López, R.Marticorena. Soporte de métricas con Independencia del Lenguaje para la inferencia de Refactorizaciones.
- [11] L. Fernández, P. J. Lara. Proceso y herramientas para la productividad en el asesoramiento y medición de calidad en desarrollos Java. Revista de Procesos y Métricas de las Tecnologías de la Información (RPM) ISSN: 1698-2029 VOL. 1, Nº 2, Agosto 2004, 31-41.
- [12] M. Rodríguez, M. Genero, J. Garzás, M. Piattini de Kybele Consulting S.L.; Madrid, España; Kybele Research, Dpto. de Lenguajes II, Universidad Rey Juan Carlos; Madrid, España, Grupo ALARCOS, Dpto. de Tecnologías y Sistemas de Información; Universidad de Castilla-La Mancha; Ciudad Real, España. Kemis: Entorno para la medición de la calidad del producto software. RPM-AEMES, VOL. 4, Nº Especial, Octubre 2007. ISSN: 1698-2029.

Páginas web:

- SpoonEMF2. <http://soft.vub.ac.be/soft/research/mdd/spoonemf2>
- Spoon. <http://spoon.gforge.inria.fr/Spoon/HomePage>
- Introduction to JET. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html
- Tutorial: Create code in Eclipse with JET.
<http://www.ibm.com/developerworks/opensource/library/os-ecl-jet/>
- Java Emitter Template Tutorial. <http://www.vogella.de/articles/EclipseJET/article.html>
- Eclipse Documentation - Standard JET2 Java Tagss.
<http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.jet.doc/references/taglibs/javaTags/resourceTag.html>
- Navigate an XMI Model with JET. [http://wiki.eclipse.org/M2T-JET-FAQ/How do I navigate an XMI model with JET%3F](http://wiki.eclipse.org/M2T-JET-FAQ/How_do_I_navigate_an_XMI_model_with_JET%3F)
- Eclipse Documentation – EMF Developer Guide.
<http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.emf.doc/references/javadoc/org.eclipse/emf/ecore/EReference.html>
- EOS. <http://maude.sip.ucm.es/eos/v0.3/doc/index.html>
- Calidad del Software. <http://www.infor.uva.es/~manso/calidad/metricasoo-2007.pdf>
- Lack of Cohesion in Methods: <http://eclipse-metrics.sourceforge.net/descriptions/LackOfCohesionInMethods.html>
- JSCS. <http://jscs.sourceforge.net/>
- CheckStyle. <http://checkstyle.sourceforge.net/>
- JavaNCSS. <http://javancss.codehaus.org/>
- Java Measurement Tool. <http://www-ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/jmt.html>
- Metrics 1.3.6. <http://metrics.sourceforge.net/>
- RSM Metrics.
http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics.htm#File%20metrics
- SDMetrics. <http://www.sdmetrics.com/index.html>
- SONAR. <http://www.sonarsource.org/>

Ilustraciones

ILUSTRACIÓN 1: OBJETIVOS: FASES DEL PROYECTO.	2
ILUSTRACIÓN 2: ESTADO DEL ARTE: HERRAMIENTA JCSC.	13
ILUSTRACIÓN 3: ESTADO DEL ARTE: HERRAMIENTA JCSC – CONFIGURACIÓN DE MÉTRICAS.	14
ILUSTRACIÓN 4: ESTADO DEL ARTE: HERRAMIENTA JAVANCSS – MÉTRICAS POR PAQUETE.	16
ILUSTRACIÓN 5: ESTADO DEL ARTE: HERRAMIENTA JAVANCSS – MÉTRICAS POR CLASE.	16
ILUSTRACIÓN 6: ESTADO DEL ARTE: HERRAMIENTA JAVANCSS – MÉTRICAS POR MÉTODO.	16
ILUSTRACIÓN 7: ESTADO DEL ARTE: HERRAMIENTA JMT.	18
ILUSTRACIÓN 8: ESTADO DEL ARTE: ECLIPSE PLUGIN METRICS 1.3.6 - PREFERENCIAS	19
ILUSTRACIÓN 9: ESTADO DEL ARTE: ECLIPSE PLUGIN METRICS 1.3.6 - CONFIGURACIÓN	20
ILUSTRACIÓN 10: ESTADO DEL ARTE: ECLIPSE PLUGIN METRICS 1.3.6 - RESULTADOS	20
ILUSTRACIÓN 11: ESTADO DEL ARTE: HERRAMIENTA RSM	21
ILUSTRACIÓN 12: ESTADO DEL ARTE: HERRAMIENTA SDMETRICS.	22
ILUSTRACIÓN 13: ESTADO DEL ARTE: HERRAMIENTA SONAR – ÁMBITOS DE EVALUACIÓN.	23
ILUSTRACIÓN 14: ESTADO DEL ARTE: HERRAMIENTA SONAR – LISTA PROYECTOS.	23
ILUSTRACIÓN 15: ESTADO DEL ARTE: HERRAMIENTA SONAR – EVALUACIÓN CHECKSTYLE.	24
ILUSTRACIÓN 16: ESTADO DEL ARTE: HERRAMIENTA SONAR – EVALUACIÓN JFREECHART.	24
ILUSTRACIÓN 17: DISEÑO TÉCNICO: FLUJO DE LA APLICACIÓN.	27
ILUSTRACIÓN 18: DISEÑO TÉCNICO: PLUGIN ES.UCM.SPOON.CLIENT	28
ILUSTRACIÓN 19: DISEÑO TÉCNICO: ESTRUCTURA ES.UCM.SPOON.CLIENT	28
ILUSTRACIÓN 20: DISEÑO TÉCNICO: DEFINICIÓN DEL PLUGIN ES.UCM.SPOON.CLIENT EN PLUGIN.XML	29
ILUSTRACIÓN 21: DISEÑO TÉCNICO: ES.UCM.SPOON.CLIENT INTRODUCE THE JAVA PROJECT PATH.	29
ILUSTRACIÓN 22: DISEÑO TÉCNICO: ES.UCM.SPOON.CLIENT METAMODEL SAVED SUCCESSFULLY.	30
ILUSTRACIÓN 23: DISEÑO TÉCNICO: ESTRUCTURA SPOONTOEOS.	30
ILUSTRACIÓN 24: DISEÑO TÉCNICO: PROPIEDADES JET DE SPOONTOEOS.	31
ILUSTRACIÓN 25: DISEÑO TÉCNICO: PESTAÑA GENERAL.	42
ILUSTRACIÓN 26: DISEÑO TÉCNICO: PESTAÑA PACKAGE, SELECCIONAR UN PAQUETE.	43
ILUSTRACIÓN 27: DISEÑO TÉCNICO: PESTAÑA PACKAGE, EVALUAR MÉTRICAS.	43
ILUSTRACIÓN 28: DISEÑO TÉCNICO: PESTAÑA CLASS, SELECCIONAR UNA CLASE.	44
ILUSTRACIÓN 29: DISEÑO TÉCNICO: PESTAÑA CLASS, EVALUAR MÉTRICAS.	44
ILUSTRACIÓN 30: DISEÑO TÉCNICO: PESTAÑA INTERFACE, SELECCIONAR UNA INTERFAZ.	45
ILUSTRACIÓN 31: DISEÑO TÉCNICO: PESTAÑA INTERFACE, EVALUAR MÉTRICAS.	45
ILUSTRACIÓN 32: DISEÑO TÉCNICO: PESTAÑA CUSTOM QUERY, EVALUAR MÉTRICA GENERAL.	46
ILUSTRACIÓN 33: DISEÑO TÉCNICO: PESTAÑA CUSTOM QUERY, EVALUAR MÉTRICA POR PAQUETE.	47
ILUSTRACIÓN 34: DISEÑO TÉCNICO: PESTAÑA CUSTOM QUERY, EVALUAR MÉTRICA POR CLASE.	48
ILUSTRACIÓN 35: DISEÑO TÉCNICO: PESTAÑA CUSTOM QUERY, EVALUAR MÉTRICA POR INTERFAZ.	49
ILUSTRACIÓN 36: RESULTADOS OBTENIDOS: ESTRUCTURA PROJECTFORTTEST.	51
ILUSTRACIÓN 37: RESULTADOS OBTENIDOS: PROJECTFORTTEST - MÉTRICAS GENERALES.	52
ILUSTRACIÓN 38: RESULTADOS OBTENIDOS: PROJECTFORTTEST – INTERFAZ NO IMPLEMENTADA.	52
ILUSTRACIÓN 39: RESULTADOS OBTENIDOS: MOTOR – EVALUACIÓN DE UNA CLASE.	53
ILUSTRACIÓN 40: RESULTADOS OBTENIDOS: JAVACOOK_v2 – EVALUACIÓN DE UNA PAQUETE.	53
ILUSTRACIÓN 41: RESULTADOS OBTENIDOS: INSERTAR NUEVA MÉTRICA.	55
ILUSTRACIÓN 42: RESULTADOS OBTENIDOS: NUEVA MÉTRICA DISPONIBLE.	56
ILUSTRACIÓN 43: APÉNDICE A: CONFIGURACIÓN SPOONJDT.	61
ILUSTRACIÓN 44: APÉNDICE B: ESTRUCTURA SPOONCLIENT.	63
ILUSTRACIÓN 45: APÉNDICE B: EJECUCIÓN SPOONCLIENT.	64
ILUSTRACIÓN 46: APÉNDICE B: ESTRUCTURA EJEMPLO 1.	64

ILUSTRACIÓN 47: APÉNDICE B: RESULTADO EJECUCIÓN SPOONCLIENT EJEMPLO 1.....	65
ILUSTRACIÓN 48: APÉNDICE B: ESTRUCTURA EJEMPLO 2.	65
ILUSTRACIÓN 49: APÉNDICE B: RESULTADO EJECUCIÓN SPOONCLIENT EJEMPLO 2.....	65
ILUSTRACIÓN 50: APÉNDICE B: ESTRUCTURA EJEMPLO 3.	65
ILUSTRACIÓN 51: APÉNDICE B: ERRORES EJECUCIÓN SPOONCLIENT EJEMPLO 3.	66
ILUSTRACIÓN 52: APÉNDICE B: RESULTADO EJECUCIÓN SPOONCLIENT EJEMPLO 3.....	66
ILUSTRACIÓN 53: APÉNDICE B: ESTRUCTURA EJEMPLO 4.	66
ILUSTRACIÓN 54: APÉNDICE B: RESULTADO EJECUCIÓN SPOONCLIENT EJEMPLO 4.....	67
ILUSTRACIÓN 55: APÉNDICE D: GENERACIÓN CÓDIGO FUENTE SPOONToEOS.....	72
ILUSTRACIÓN 56: APÉNDICE D: EJECUCIÓN SPOONToEOS.....	72

Tablas

TABLA 1: DISEÑO TÉCNICO: MÉTRICAS GENERALES.	38
TABLA 2: DISEÑO TÉCNICO: MÉTRICAS POR PAQUETE.	39
TABLA 3: DISEÑO TÉCNICO: MÉTRICAS POR CLASE.	41
TABLA 4: DISEÑO TÉCNICO: MÉTRICAS POR INTERFAZ.	42
TABLA 5: RESULTADOS OBTENIDOS: TIEMPOS DE RESPUESTA.	54
TABLA 6: RESULTADOS OBTENIDOS: COMPARATIVA ENTRE HERRAMIENTAS.....	55